NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION
OF
A PERSONNEL DATABASE

by

Bora BUYUKONER
and
Yucel OZIN

June 1985

Thesis Advisor: Samuel H. Parry

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: Design and Implementation of a Personnel Database	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis June 1985
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Buyukoner, Bora; Yucel, Ozin	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	12. REPORT DATE: June 1985
	13. NUMBER OF PAGES: 156
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Database, Personnel Database, Database Design, Database Models, DBMS, Normal Forms

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis deals with the design considerations for a personnel database system. It introduces the important concepts related to the analysis and design phases of a database system. Two types of data models, namely conceptual and implementation models, are described, particularly concentrating on the Semantic Data Model for implementation. The Semantic Data Model is used to indicate

(Continued)

ABSTRACT (Continued)

the entities and relationships between those entities for the Personnel Database. After the completion of this process, the SDM design is converted into a corresponding relational database which is implemented using the ORACLE Database Management System (DBMS).

Approved for public release; distribution is unlimited.

Design and Implementation of a Personnel Database

by

Bora BUYUKONER
Major, Turkish Army
B.S., Military Academy, Ankara/TURKEY, 1965

and

Yucel OZIN
Captain, Turkish Army
B.S., Military Academy, Ankara/TURKEY, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

ABSTRACT

This thesis deals with the design considerations for a personnel database system. It introduces the important concepts related to the analysis and design phases of a database system. Two types of data models, namely conceptual and implementation models, are described, particularly concentrating on the Semantic Data Model (SDM) for conceptual and the Relational Data Model for implementation. The Semantic Data Model is used to indicate the entities and relationships between those entities for the Personnel Database. After the completion of this process, the SDM design is converted into a corresponding relational database which is implemented using the ORACLE Database Management System (DBMS).

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Around 1964 a new term appeared in the computer literature to denote a new concept. The term was "database," and it was going to play a highly significant role in an organization's information system. The information system supports the organization's functions, maintaining the data for these functions and assisting users to interpret the data for decision making. The database becomes an important tool in this process; it is the container of the data in the information system.

In many information systems, database denotes collections of data shared by end-users of computer systems. Users who make decisions obtain data by accessing the database and then recording their decision in it. Easy access to a variety of data from a number of locations enables the information system to quickly respond to the needs of decision makers within the organization, whereas poor access can of course hinder rapid response. If the data are not readily available, decisions may be either delayed unnecessarily or made with incomplete data, leading to possible system malfunction in the future.

The flexibility of the database structures is a very important feature to meet changing organizational needs. As new functions arise in an organization, new decisions follow in their wake. Since the database will need to store new data and accommodate new relationships to support the new decisions, it must include facilities to allow such changes to be easily made. [Ref. 1:pp. 1-3]

Today, computer applications in which many users at terminals concurrently access a database are called "database applications" [Ref. 2]. A significant new kind of

software, the database management system, or DBMS, has evolved to facilitate the development of database applications. The development of DBMS, in turn, has given rise to new languages, algorithms, and software techniques which together make up what might be called a database technology.

Database technology has been driven by, and to a large extent distinguished from other software technologies by the following broad user requirements.

- Data consolidation
- Data independence
- Data protection

In the years ahead, database systems will become increasingly widespread and increasingly important. At present, however, they represent a new and relatively unexplored field, despite the fact that the number of systems installed or under development is growing rapidly.

The primary goal of this thesis is to present the design steps of a particular database system, design criteria, and the elements of the database system which provide designers with the ability to evaluate databases against these criteria. The second objective of this thesis is to show the implementation of that database system which controls and executes the transactions written in a model-based database language such as Data Definition Language (DDL) and Data Manipulation Language (DML). Finally, the third objective of this study is to introduce essential features of the maintainability, administration, and security of a database management system.

Chapter II describes the basic concepts of database, including the definition of a DBS, its components, its architecture, and some advantages/disadvantages. Chapter III briefly reviews the design objectives and techniques of a database and describes logical and physical database design. Chapter IV also briefly addresses database models which can

be used to form a logical framework of a database and to support further design phases and/or to create intended database structure which will be implemented after the completion of design phases. Chapter V introduces, in detail, the Semantic Database Model for a personnel assignment database. Chapters VI and VII describe the design of the personnel database by using the Relational Database Model approach which is one of the three database models. In addition rules, design criteria, and important operations associated with this model are given. Chapter VII also shows how the designer can transform the SDM model which has been designed for a personnel database system into a relational database model. The INGRES Database Management System which is available today is discussed in Chapter VIII. Chapter IX demonstrates the implementation of the relational database system which is implemented on the VAX computer systems by using the ORACLE Relational DBMS. Chapter X describes the functions of a DBMS, such as security features, maintainability, and concurrent processing control. Finally, conclusions and recommendations based on our research are presented in Chapter XI.

II. BASIC CONCEPTS OF DATABASE

A. DEFINITION OF A DATABASE SYSTEM

The simplest definition of a database might be that a database is a collection of facts or a repository for stored data which is both integrated and shared. By "integrated" we mean that the database may be considered as a unification of several otherwise distinct data files, with any redundancy among those files partially or wholly eliminated. By "shared" we mean that individual pieces of data in the database may be shared among several different users, in the sense that each of those users may have access to the same piece of data. The term "shared" is also extended to cover concurrent sharing: that is, the ability for several users to be accessing the database at the same time. [Ref. 3:pp. 3-7]

R.W. Engles [Ref. 4] refers to the data in a database as "operational data," distinguishing it from input data, output data, and other kinds of data. Thus, a modified version of Engles' original definition of database is that a database is a collection of stored operational data used by the application systems of some particular enterprise. "Enterprise" is simply a convenient generic term for any reasonably self-contained commercial, scientific, technical, or other organization. Any enterprise must necessarily maintain a large amount of data about its operation. This is its "operational data," such as product data, account data, military personnel data etc.

In recent years, technology improved to the point where it became feasible to design, build, and operate large-scale collections of data in a computer environment. In other

words, organizations realized that data were a valuable resource and needed to be centrally managed. The concept of a database has thus emerged fully only in recent years. A database can also be defined as a computerized collection of stored operational data that serves the needs of multiple users within one or more organizations [Ref. 5:pp. 3-17]. A key point is that the database is an integrated resource to be used by all members of the organizations who need information contained in it.

Since the database is an integrated and shared resource for multiple users within an organization, it should be managed for the organization's benefit and from its viewpoint, not by individual users. Thus, two additional concepts have been developed to solve the problem of controlling and managing the organization's database resource. Initially, software was developed to provide a common interface between all users and the integrated database. A common interface promotes privacy and data integrity. Also, users cannot store information implicitly and must use and modify data in a manner consistent with the organization's viewpoint. The software, known as a database management system, allows computer control of the data resource. A database management system (DBMS) is a collection of software tools and access methods which enables the users to store facts about real-world objects and the relationships between these objects, and to manipulate those facts by issuing queries in content-addressable form. In short, a DBMS is a generalized tool for manipulating a database [Ref. 5:pp. 3-17] ; it is made available through special software for the interrogation, maintenance, and analysis of data.

The second concept is that of the database administrator (DBA). The DBA can be thought of as one or more individuals, possibly aided by a staff, who manage the organization's

database resource [Ref. 5:pp. 3-17], or are responsible for overall control of the database system. The DBA's responsibilities include the following [Ref. 3:pp. 25-26].

- .Deciding the information content of the database.
- .Deciding the storage structure and access strategy.
- .Liaising with users.
- .Defining authorization checks and validation procedures.
- .Defining a strategy for backup and recovery.
- .Monitoring performance and responding to changes in requirements.

We can clearly see the reason why an organization should choose to store its operational data in an integrated database. A database system provides the organization with centralized control of its operational data which is one of its most valuable assets. This is in sharp contrast to the situation that prevails in many organizations today, where typically each application has its own private files so that the operational data is widely dispersed, and is therefore probably difficult to control.

B. COMPONENTS OF A DATABASE SYSTEM

A database system consists of four major components: data, hardware, software, and users. Fig 2.1 shows a greatly simplified view of the major components of a database system.

1. Data

The data stored in the system is partitioned into one or more databases. For tutorial purposes it is usually convenient to assume that there is just one database, containing the totality of all stored data in the system.

According to standard usage in the computer industry, bits are grouped into bytes or characters,

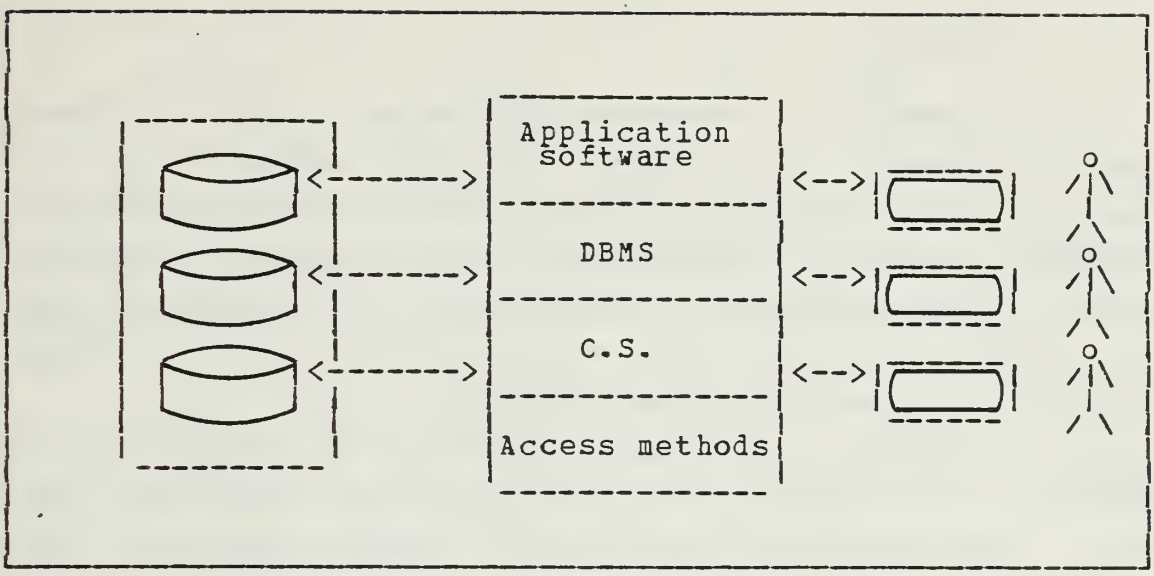


Figure 2.1 Simplified View of a Database System.

characters are grouped into fields, and fields are grouped into records. A collection of records is called a file. At this point, we cannot say that a database is a collection of files. A database is a collection of "integrated" files and relationships among records in those files. Database processing differs from file processing in which the structure of the files is distributed across the application programs and each file is considered to exist independently. On the other hand, the database is self-describing because it contains, within itself, a description of its structure. Another difference between file processing and database concerns the term file. For file processing, the records in a file are usually grouped together physically. For database processing, the logical collection of records probably does not exist as a physical collection. In database processing, there are logical files, or collections of records having meaning to users, and physical files, or collections of records on physical devices.

2. Hardware

In general, database applications do not require special hardware. It consists of direct access storage (or secondary storage) devices (disks, drums, etc.) on which the database resides, together with the associated devices, control units, channels, and so forth. It is assumed that the database is too large to be stored in its entirety within the computer's primary storage.

In 1982, a new term appeared and several vendors announced new products called "database machines" [Ref. 6:p. 8]. These machines are special purpose computers that perform database processing functions. According to this type of architecture, the main frame or host computer sends requests for service and data over a channel to the database machine. The machine processes the requests and sends results, messages, or data back to the main computer. Thus database processing can be performed simultaneously with applications processing. The actual effectiveness of such machines is under investigation. If substantial processing efficiencies can be proved at a reasonable cost, then database machines will become important. Hardware aspects of the system form a major topic in their own right; the problems encountered in this area are not peculiar to database systems, and those problems have been very thoroughly investigated and documented elsewhere. Thus, this thesis is not concerned with hardware aspects of the system.

3. Software

The database management system or DBMS is a layer of software which provides the interface between the physical database itself (i.e., the data as actually stored) and the users of the system. All requests from users for access to the database are handled by the DBMS. One general function

provided by the DBMS is the separation of database users from hardware-level detail. In other words, the DBMS provides a view of the database that is elevated somewhat above the hardware level, and supports user operations (such as "get the OFFICER record for officer Buyukoner") that are expressed in terms of that higher-level view. This function, and other functions of the DBMS, will be discussed in detail later.

Two types of programs involved in database processing are the Operating System (OS) and Communications Control Program (CCP). The operating system is a set of programs which controls the computer's resources. In a sense, the OS can be viewed as the glue that holds all of the other programs together. Communications control program (CCP) performs communications-oriented tasks. On-line processing requests or transactions are provided by users at terminals. The requests are received and routed by the CCP over communications lines. The CCP has several important functions: provides communications error detection and correction, manages terminal activity, routes messages to the correct next destination, and formats messages for various types of terminal equipment. The CCP also routes on-line input to the next level of programs which contains application programs and database utilities. The operating system and the CCP will not be discussed further in this thesis.

4. Users

There are three broad classes of user being considered: application programmers, end-users, and the database administrator (DBA). [Ref. 3:p. 6]

The application programmer is responsible for writing application programs that use the database, typically in a high-level language such as COBOL or PL/I. These

application programs are used with the data for retrieving information, creating new information, and deleting or changing existing information. The programs themselves may be conventional batch applications, or they may be "on-line" programs that are designed to support an end-user interacting with the system from an on-line terminal.

The end-user can access the database from a terminal. An end-user may, in general, perform all the functions of retrieval, creation, deletion, and modification by employing a query language provided as an integral part of the system, or by invoking a user-written application program that accepts commands from the terminal and in turn issues requests to the DBMS on the end-user's behalf.

The database administrator, or DBA mentioned earlier in this Chapter, is the person (or group of persons) responsible for overall control of the database system. The function of the DBA staff is to serve as a protector of the database and as a focal point for resolving users' conflicts.

C. ADVANTAGES AND DISADVANTAGES OF DATABASE PROCESSING

The main advantage of database processing is included in its definition given previously. Integrated and shared data offers those important advantages. Database processing allows more information to be produced from a given amount of data. Secondly, the amount of redundancy in stored data can be minimized. In other words, the elimination or reduction of data duplication allows data to only be stored once. As a result, this saves file space, and to some extent, can reduce processing requirements. [Ref. 6:pp. 3-8], and [Ref. 7:pp. 1-16]

As mentioned earlier, centralized control of the operational data in a database provides the following advantages [Ref. 3:pp. 10-12].

1. Avoidance of Inconsistency

This is really a corollary of the above point. If a given fact about the real world is represented by two different entries in the database and the redundancy is not controlled, then there will be some occasions for which the two entries will not agree (that is, when only one has been updated). At such times the database is said to be inconsistent. In this case, the database produces incorrect or conflicting information. If the redundancy is controlled, then the system could guarantee that the database is never inconsistent as seen by the user, by ensuring that any change made to either of the two entries is automatically made to the other. This process is known as propagating updates (the term "update" is used to cover all the operations of creation, deletion, and modification).

2. Shared Data

The concept of shared data was discussed in Section A.

3. Enforcement of Standards

The applicable standards, which may include any or all of the following: installation, company, industry, and national standards, are followed in the representation of the data. Standardizing stored data formats assists in data interchange or migration between systems.

4. Application of Security Restrictions

The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted (see Chapter X for more detail).

5. Maintenance of Data

The problem of integrity is the problem of ensuring that the data in the database is accurate. Inconsistency between two entries representing the same "fact" leads to a lack of data integrity (which can occur only if redundancy exists in the stored data). It is essential to point out that data integrity is even more important in a database system than in a "private files" environment, because the database is shared. Centralized control of the database supports data integrity.

6. Balancing of Conflicting Requirements

Knowing the overall requirements of the enterprise, the DBA can structure the database system to provide an overall service that is "best for the enterprise."

The cost of database processing may become a major disadvantage. It can be expensive. The DBMS may need so much primary storage that additional storage must be purchased. Even with more storage, it may get exclusive control of the CPU, thus forcing the user to upgrade to a more powerful computer. [Ref. 6:pp. 3-8]

Once the database is implemented, operating costs for some systems will be higher. For example, sequential processing will never be done as fast in the database environment, since it causes excessive overhead.

Large amounts of data in different formats can be interrelated in the database. Both the database system and the application programs must be able to process these structures. This requires more sophisticated programming, takes time, and requires highly skilled programming personnel. Thus, the complexity is another important disadvantage of database processing. Backup and recovery also increases complexity and are more difficult in the database

environment to carry out. Another reason for this is that databases are often processed by multiple users concurrently. Determining the exact state of the database at the time of failure may be a problem. Given that, it may be even more difficult to determine what should be done next.

Another disadvantage is that integration, and hence centralization, increases vulnerability. A failure in one component of an integrated system can cause the entire system to fail. This event is especially critical if the operation of the user organization depends on the database.

To avoid these potential drawbacks a database management system (DBMS) should satisfy the following objectives [Ref. 7:pp. 13-14] :

- Different functions of an enterprise can be served effectively by the same DBMS.
- Redundancy in stored data can be minimized.
- Consistent information can be supplied for the decision-making process.
- Security controls can be applied.
- Application programs can be developed, maintained, and enhanced faster and more economically, with fewer skilled personnel.
- Physical reorganization of the stored data is easy.
- Centralized control of the database is possible.
- Easier procedures for computer operations can be established.

D. AN ARCHITECTURE FOR A DATABASE SYSTEM

An architecture for a database system is illustrated in Fig. 2.2 [Ref. 3:p. 20]. This picture presents a framework which is extremely useful for describing general database concepts and for explaining the structure of individual systems, and it is in broad agreement with that proposed by

the ANSI/SPARC Study Group on Data Base Management Systems [Ref. 8].

The architecture is divided into three general levels: internal, conceptual, and external. Generally speaking, the external level is the one closest to the users; that is, the one concerned with the way in which the data is viewed by individual users. The internal level is the level closest to physical storage; that is, the one concerned with the way in which the data are actually stored. The conceptual level is a bridge or "level of indirection" between the other two. There may be many "external views," each consisting of a more or less user oriented logical representation of some portion of the database (such as logical records and fields), and there may be a single "conceptual view," consisting of a similarly logical representation of the entire database. Likewise, there will be a single "internal view," representing the total database as actually stored.

The three levels are also defined as levels of abstraction and named in the specification of a database structure: the conceptual or enterprise administrator view, the implementation view of the applications programmer or end user, and the physical view of the systems programmer/analyst [Ref. 5:pp. 3-17]. The external level, conceptual level, and internal level in the ANSI/SPARC model correspond to the implementation level, conceptual level, and physical level in the levels of abstraction, respectively. Figure 2.3 shows these three levels of abstraction and some of their primary components.

It should be obvious that between the computer, dealing with bits, and the ultimate user, dealing with abstractions such as military units or assignment of personnel to a division, there will be many levels of abstraction. It should be emphasized that only the database actually exists at the physical level. We may view the physical database itself at

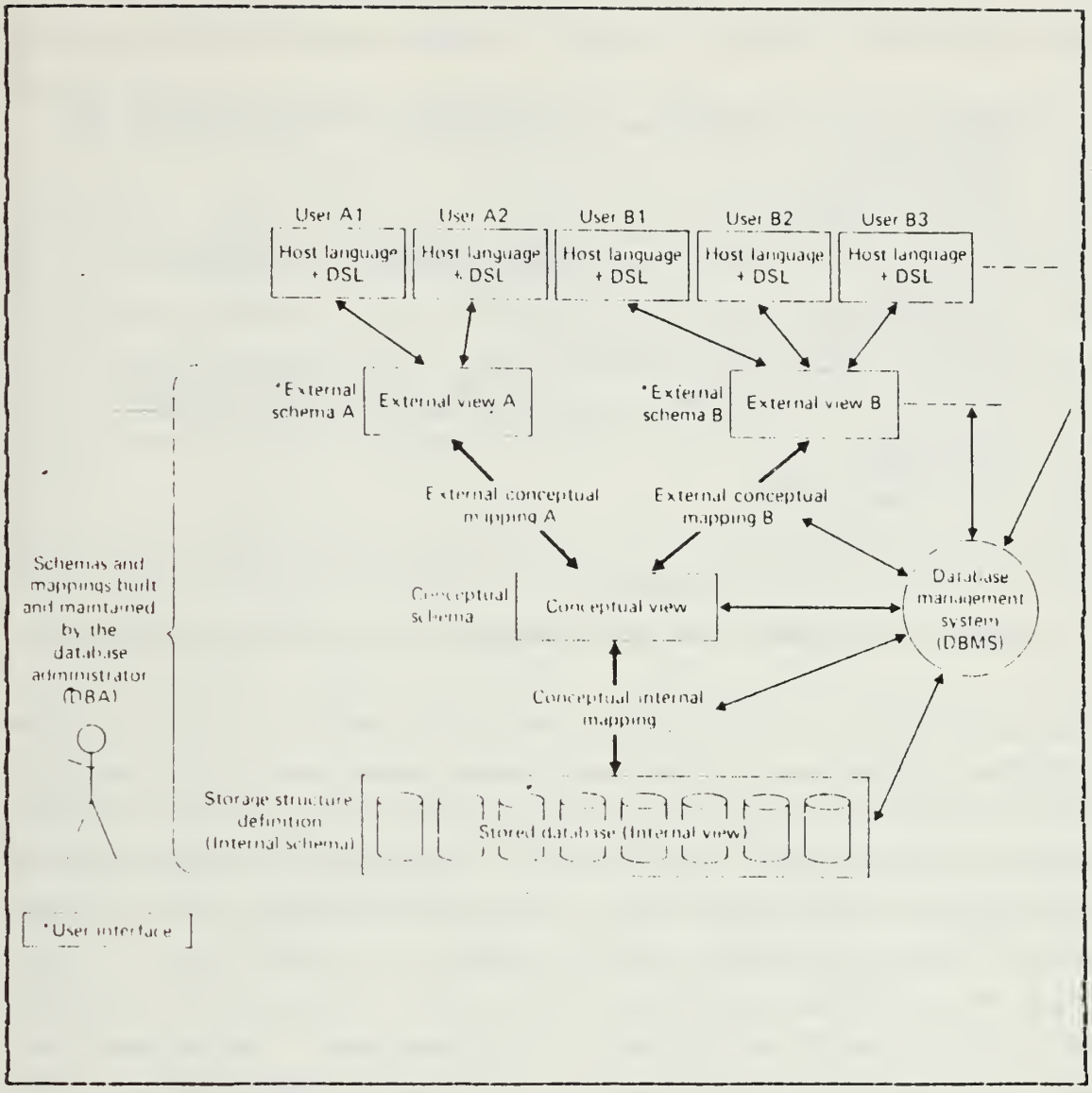


Figure 2.2 Database System Architecture.

several levels of abstraction, ranging from that of records and files in a programming language such as Pascal, through the level of logical records, as supported by the operating system underlying the DBMS, down to the level of bits and physical addresses on storage devices. We may also view the conceptual database as an abstraction of the real world pertinent to an enterprise. J.D. Ullman [Ref. 9:pp. 5-9]

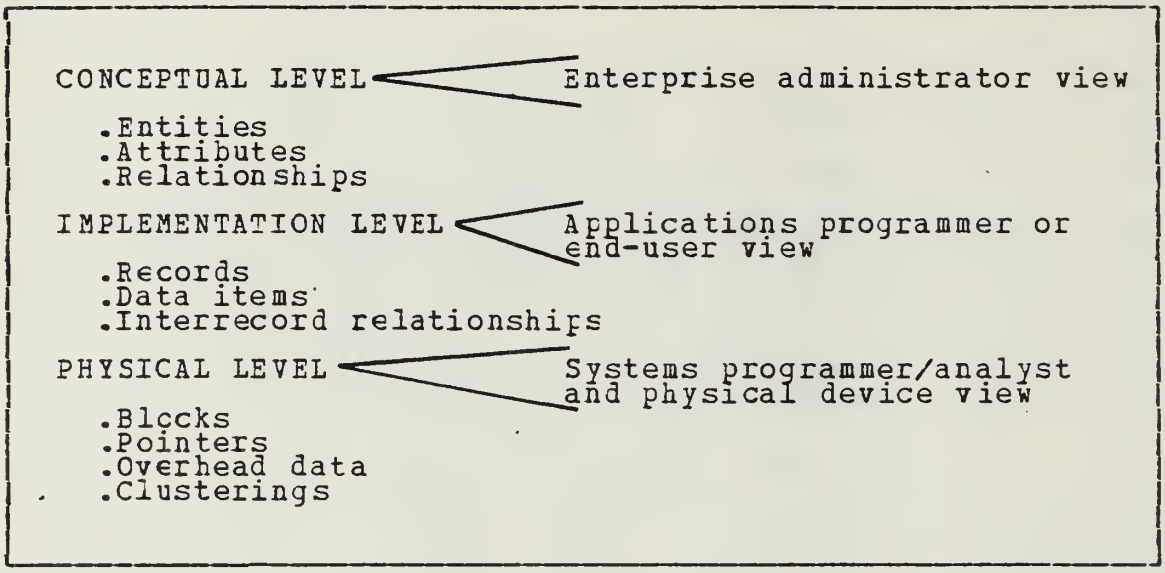


Figure 2.3 **Levels of Abstraction in a Database System.**

emphasizes that a view (or external view) is an abstract model of a portion of the conceptual database. As an example of the utility of views, the army may provide a computerized personnel assignment department, consisting of data and a collection of programs that deal with officers and military units. These programs, and the people who use them, do not require knowledge concerning personnel files or the assignment of officers to units. The personnel department may need to know about assignments, units, and aspects of the personnel files (e.g., which officers are qualified to assign to unit X), but does not need to know about personnel salaries. Thus, there may be one view of the database for the personnel department and another for the finance department.

> "In a sense, a view is just a small conceptual database, and it is at the same level of abstraction as the conceptual database. However, there are senses in which a view can be "more abstract" than a conceptual database, as the data dealt with by a view may be constructable from the conceptual database but not actually present in that database." [Ref. 9:p. 7]

Figure 2.2 illustrates the several components of the architecture in more detail. Next, we will examine these components.

Each user has a language at his or her disposal. For the application programmer it will be a high-level programming language, such as PI/I or COBOL; for the terminal user it will be either a query language or a special-purpose language supported by an on-line application program to meet the user's requirements. Each of those languages is known as "host language." The term "data sublanguage (DSL)" is a subset of the host language that is concerned with database objects and operations. In other words, the DSL is embedded in a host language. Multiple host languages and multiple DSLs may be supported by a given system. In principle, any given data sublanguage is really a combination of two languages [Ref. 3:pp. 17-25], a data definition language (DDL), which provides for the definition or description of database objects, and a data manipulation language (DML), which supports the manipulation or processing of such objects. In most systems today the data sublanguage and the host are very loosely coupled. That is, the definitions written in DDL are completely outside the application program.

An external view is the content of the database as it is seen by some particular user. In general, an external view consists of multiple occurrences of multiple types of external records [Ref. 3:pp. 17-25]. An external record refers to a "logical record" which is not necessarily the same as a stored record (see Section E of this Chapter).

Each external view is defined by means of an external schema, which is made up of definitions of each of the different types of external records in that view. The term "view" is used for a set of record occurrences and the term "schema" is used for the definition of that view. The DDL

portion of the data sublanguage is used to write the external schema. That DDL is sometimes called an external DDL.

The conceptual view is a representation of the entire information content of the database in a form that is somewhat abstract in comparison with the way in which the data are physically stored. The conceptual view is composed of multiple occurrences of multiple types of conceptual records. It is more desirable to consider "entities," and "relationships" instead of dealing in terms of "conceptual records." A conceptual record is not the same as either an external record or a stored record. For example, the conceptual view may consist of a collection of branch record occurrences plus a collection of military personnel record occurrences plus a collection of course record occurrences, and so on. The conceptual view is defined by means of the conceptual schema, which includes definitions of each of the several types of conceptual records. The conceptual view is a view of the total database content, and the conceptual schema is a definition of this view. The conceptual schema is written using another DDL called conceptual DDL. It is intended that the definitions in the conceptual schema include many additional features, such as the authorization checks and validation procedures, and these definitions must not involve any considerations of storage structure or access strategy. In other words, there must not be any reference to stored field representations, physical sequence, hash-addressing, indexing, or any other storage/access details. At this level, the situation allows the conceptual model to be "data independent" which will be discussed in the next Section.

Some authorities would suggest that the fundamental objective of the conceptual schema is to describe the entire enterprise; not just its operational data, but also how that

data is used, how the data flows within the enterprise, what the data is used for at each point, what audit controls are to be applied at each point, and so on.

The internal view or physical level of the architecture is a very low-level representation of the entire database. It consists of multiple occurrences of multiple types of stored records (the ANSI/SPARC refers to this term as "internal record"). The internal view is defined by means of the internal schema, which not only defines the various types of stored records but also specifies what indexes exist, how stored fields are represented, what physical sequence the stored records are in, and so on [Ref. 3:pp. 17-25]. Another data definition language (the internal DDL) is used to write the internal schema. It is convenient to use the term "stored database" in place of "internal view," and "storage structure definition" in place of "internal schema."

Two levels of mapping are shown in Fig. 2.2 The conceptual/internal mapping describes the correspondence between the conceptual view (or data model) and the stored database; it specifies how conceptual records and fields map into their stored counterparts. If the structure of the stored database is changed, the conceptual/internal mapping must be changed accordingly, so that the conceptual schema may remain invariant. For example, if a change is made to the storage structure definition of the database, the effects of such a change must be contained below the conceptual level, so that "data independence" can be accomplished.

An external/conceptual mapping describes the correspondence between a specific external view and the conceptual view. In general, the same kind of differences may exist between these two levels as may exist between the conceptual view and the stored database. For example, records may be in different sequences, fields may have different data types,

and so on. Different external views may overlap. That is, any number of external views may exist at the same time and any number of users may share a given external view. Some systems allow the definition of one external view to be expressed in terms of others without always requiring an explicit definition of the mapping to the conceptual level. If various external views are strictly related to one another, this will be a very useful feature of the system.

Referring again to Fig. 2.2, there still remain three components of the architecture: the database management system (DBMS), the database administrator (DBA), and the user interface. The DBMS is the software that handles all access to the database. The basic steps that occur in a DBMS are the following [Ref. 3:pp. 17-25] :

1. A user issues an access request, using some particular data manipulation language,
2. the DBMS intercepts the request and interprets it,
3. the DBMS inspects, in turn, the external schema, the external/conceptual mapping, the conceptual schema, the conceptual/internal mapping, and the storage structure definition, and
4. the DBMS performs the necessary operations on the stored database.

For example, assume that a user wishes to retrieve a particular external record occurrence. In general, the DBMS must retrieve all required stored record occurrences, construct the required conceptual record occurrences, and then construct the required external record occurrence. At each step, data type or other conversions may be necessary. Whenever a retrieval request occurs, fields will be required from several conceptual record occurrences, and each conceptual record occurrence, in turn, may require fields from several stored record occurrences.

The database administrator (DBA), previously discussed to some extent, controls the overall database system. We will only mention the utilities and tools which are required to achieve the DBA's tasks. Such utilities would be an essential part of a database system. For instance, loading routines, reorganization routines, journaling routines, recovery routines, and statistical analysis routines may be included as utilities. One of the most important DBA tools is the "data dictionary" (not shown in Fig. 2.2). The data dictionary is effectively a database in its own right (that is, descriptions of other objects in the system). In particular, all the various schemas (external, conceptual, internal) are physically stored in both source and object form in the dictionary. A comprehensive dictionary will also include cross-reference information, showing, for example, which programs use which pieces of data, which departments require which reports, and so on. It is possible to query the dictionary just like any other database, so that the DBA can easily discover which programs are likely to be affected by some change to the system.

A data dictionary should help a database user in the following ways: [Ref. 7:pp. 20-21]

- .Communicating with the other users.
- .Controlling the data elements in a simple and effective manner, that is, introducing new elements into the systems, or changing the definitions of the elements.
- .Reducing data redundancy and inconsistency.
- .Determining the impact of changes to data elements on the total database.
- .Centralizing the control of the data elements as an aid in database design and in expanding the design.

The user interface, shown in Fig. 2.2, may be defined as a boundary in the system below which everything is transparent (invisible) to the user. Thus, the user interface is at the external level.

E. DATA INDEPENDENCE

The concept of data independence may be easily understood by first introducing its opposite. Currently, many applications are data-dependent. This means that the requirements of the application dictate both the way in which the data are organized in secondary storage and the way in which they are accessed, and, moreover, that knowledge of the data structure and access method is built into the application logic. In this case, the application programmer has to know the data format, the location of where the data is stored, and the access method which tells how the data is accessed. Changes in any of these items may affect the application program and result in other changes, since the details of these three points may be embedded into the application code. It is also likely that as the needs of the enterprise change, the format of the data may change, and the data set has to be expanded by adding information about different types of entities or additional information about existing entities.

It is said that an application such as above is data-dependent because it is impossible to change the storage structure (how the data is physically stored) or the access method without affecting the application. For example, it would not be possible to replace an indexed sequential file by a hash-addressed file without making any changes to the application.

In a database system, there are at least two important reasons why applications must be data-independent [Ref. 3:pp. 12-17].

1. Different applications will need different views of the same data.
2. The DBA must have the freedom to change the storage structure or access strategy (or both) in response to

changing requirements without having to modify existing applications. For example, the enterprise may adopt new standards, application priorities may change, new types of storage device may become available, and so on.

If applications are data-dependent, such changes involve corresponding changes to programs, requiring programmers to spend an increasing percentage of their time in program maintenance and updating.

Therefore, it is obvious that the provision of data independence is a major objective of database systems. S. Atre [Ref. 7:p. 17] defines data independence as

> "The ability to use the database without knowing the representation details."

It can also be defined as the immunity of applications to change in storage structure and access strategy which implies that the applications concerned do not depend on any one particular storage structure and access strategy. In Section D, we have presented an architecture for a database system that provides a fundamental principle for achieving this objective.

Data independence provides, at a central location, a solution to the problems discussed above. The individual application programmer is not required to change the application programs to accommodate changes in access method or location or format of the data. Unfortunately, it is difficult to achieve full data independence in a database system, since a database design depends on the availability of the DBMS software packages today, even with the best database design. The central location for reflecting changes in the storage structure and the access strategy should be anchored in the DBMS. The important point here is when, where, why, and who should specify the changes to the DBMS, and who

should control these changes? The DBA should ,of course, be given these responsibilities.

The reasons for data independence are summarized as follows:

> "1. To allow the DBA to make changes in the content, location, representation and organization of a database without causing reprogramming of application programs which use the database.
>
> 2. To allow the supplier of data processing equipment and software to introduce new technologies without causing reprogramming of the customer's application.
>
> 3. To facilitate data sharing by allowing the same data to appear to be organized differently for different application programs.
>
> 4. To simplify application program development and, in particular, to facilitate the development of programs for interactive database processing.
>
> 5. To provide the centralization of control needed by the DBA to insure the security and integrity of the database." [Ref. 7:pp. 17-18]

The levels of abstraction, mentioned in Section D above, from the external view to conceptual to internal view, provides two stages of "data independence." In a well-designed database system, the internal schema can be modified by the DBA without altering the conceptual schema or requiring a redefinition of the external schemas (or subschemas). This independence is known as physical data independence. The advantage of physical data independence is that it permits "tuning" of the internal schema for efficiency while allowing application programs to run as if no change had occurred [Ref. 9:pp. 5-9].

The relationship between external views and the conceptual view also gives a type of independence called logical data independence. Many changes to the conceptual schema can be made without affecting existing external schemas, and other changes to the conceptual schema can be made if the external/conceptual mapping is redefined by the DBA, Again, no change to the application programs is necessary [Ref. 9:pp. 5-9].

In order to use standard terms as much as possible, it is essential to give some definitions for a database system.

> "A stored field is the smallest named unit of data stored in the database. The database, in general, contains many occurrences or instances of each of several types of stored fields.
>
> A stored record is a named collection of associated stored fields. A stored record occurrence or instance consists of a group of related stored field occurrences (and represents an association between them). In most systems, the stored record occurrence is the unit of access to the database.
>
> A stored file is the (named) collection of all occurrences of one type of stored record." [Ref. 3:p. 14]

We conclude this Chapter by pointing out that the DBMS can provide independence from:

- underlying representations such as representation of numeric data, representation of character data, data encoding/decoding, and units for numeric data.
- data structure such as materialization of computed fields, and structure of records and files.

III. AN OVERVIEW OF DATABASE DESIGN

A. INTRODUCTION

Designing a database is a difficult,complex and time-consuming process. Unfortunately, inadequate databases result because they cannot satisfy the present or future organizational requirements.

The process of developing a database structure from user requirements is called database design. Many database designers have argued that there are at least two separate steps in the database design process: the design of a logical database structure which is processible by the DBMS and describes the user's view of the data, and the selection of a physical structure that includes data representation or encoding, access methods, and physical clustering of data. Other than the logical/physical description, however, the overall structure of the design process has not been well defined, and even the logical/physical boundary has been open to considerable dispute.

General information requirements include a statement of the objectives of the database system, definition of the data elements to be included in the database, and a description of data element usage in the users' organizations. These requirements are not tied to any specific application; therefore, database structure design based on such requirements is considered to be advantageous for long-term databases that must be adaptable to changing applications.

Processing requirements consist of three distinguishable components; specific data items required for each application, the data volume (number of data occurrences), and processing frequencies in terms of the number of times each

application must be run per unit time. DBMS specifications and the operating system/hardware configuration are also used by the designer.

Performance measures and performance constraints should be considered by the designer. Typical constraints include upper bounds on response times to queries, recovery times from system crashes, or specific data needed to support certain security or integrity requirements.

Two major results of the database design process are the complete database structure and guidelines for application programmers based on database structure and processing requirements.

B. DATABASE SYSTEM LIFE CYCLE

The database system life cycle is a convenient and useful framework from which to view the database system as it evolves over time. This framework provides an ordered background to the functions of a database administrator and is divided into three separate phases: analysis and design, database operation, and reorganization. These three phases are composed of the following steps:

- Analysis and design phase
  1. Requirements formulation and analysis
  2. Conceptual design
  3. Implementation design
  4. Physical design
- Database implementation and operation phase
  1. Database implementation
  2. Operation and monitoring
- Reorganization phase (Modification and adaptation)

C. ANALYSIS AND DESIGN PHASE

A stepwise design methodology for database designer or database administratcr will be explained in this section. The general interconnections between steps are illustrated in Figure 3.1 [Ref. 5:p. 26].

1. Requirements Formulatiocn and Analysis

Requirements formulatiocn and analysis constitute the most important step cf the entire database design process, since most subsequent design decisions are based on this step. It is,however, the most poorly defined and time-consuming step of the entire process.

Contemporary database applications are very broad and very sophisticated. Many diverse applications may use the same integrated database. The design of a database to support all the applications becomes very complex. A design, without sufficient information to support the analysis, will not be valid.

The major task is collecting information content and processing requirements from all the identified and potential users of the database. Analysis of the requirements ensures the consistency of users' objectives as well as the consistency of their views of the organization's information flow.

This activity includes the establishment of organizational objectives, derivation of specific database requirements from those objectives or directly from management and nonmanagement personnel, and documentation of those requirements in a form that is aggreeable to both end users and database designers. The technique used is personal interviews with various levels of management and key employees involved in the processing of data and services in the organization. [Ref. 5:p. 25].

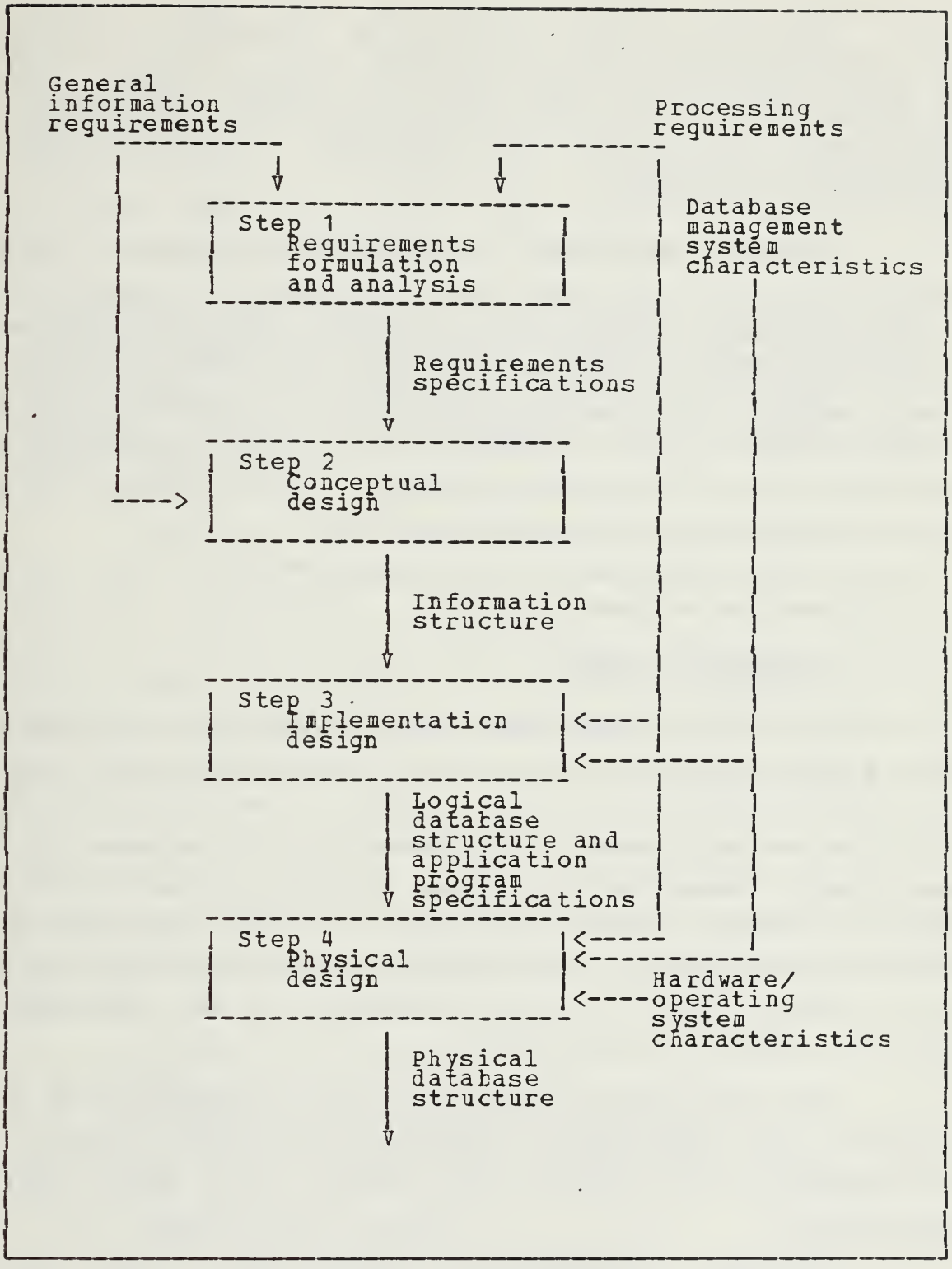


Figure 3.1 Basic Database Design Steps.

There is a need for corporate requirements analysis in the requirements formulation and analysis step. Data items and their relationships must be defined and conflicts are at least recognized, if not resolved, during corporate requirements analysis.

Different departments use different names for the same things, and the same names for different things, so that a preliminary common view of data and processes must be available before later steps can provide reliable results. Such a common view can be derived only in cooperation with users. However, this common view will not necessarily resemble the final database structure. In conclusion, there are two design constraints for this step:

.accurately modeling real world requirements

.aggregating individual views.

2. Conceptual Design

Conceptual design deals with information independent of any actual implementation (i.e. any particular hardware or software system). The main purpose of conceptual design is to represent information in a form that is comprehensible to the user independent of system specifics, but implementable on several systems. The result of conceptual design is called the conceptual schema because it is a representation of the user's "world" view and independent of any DBMS software or hardware considerations.

This step results in a high-level representation of diverse users' information requirements such as an entity-relationship (E-R) diagram or a Semantic Data Model (SDM) application.

In most representation mechanisms, the users describe their information needs in terms of entities, attributes, and relationships (E-R diagrams), or in terms of

records, items, and sets using a DBMS's data description language (DDL). It is clear that a great deal of gererality and potential design optimality are lost when the user is restricted to a particular low-level data description language instead of a higher-level representation mechanism to specify their information requirements. Similarly, the goal of the Semantic Data Model (SDM) is as follows:

> "Our goal is the design of a higher-level database model that will enable the database designer to naturally and directly incorporate more of the semantics of a database into its schema. Such a semantics-based database description and structuring formalism is intended to serve as a natural application modeling mechanism to capture and express the structure of the application environment in the structure of the database." [Ref. 10:p. 352]

There are two major reasons for a designer to use a high level of abstraction in the design process. First, entities, attributes, and relationships are not always explicitly distinguished and the design decisions are often fuzzy. Second, the problem of consistency checking would be simplified if a common, high-level information representation for conceptual information structures could be develcped.

As an example, conceptual design can be done by entity modeling which is the representation and integration of user views in terms of entity diagrams. There are four basic design decisions required to formulate the entity diagrams.

1. Selection of entities
2. Selection of entity attributes
3. Selection of key attributes for entities
4. Selection of relationships between entities.

3. Implementation Design

The major goal of the implementation design step is to use the results of the conceptual design step and the

processing requirements as input to create a DBMS-processible schema as output. Refinements to the database structure that occur during this design step are developed from the viewpoint of satisfying DBMS-dependent constraints as well as constraints specified in the user requirements.

Implementation design contains database structure design and design of programs. The database structure is a DBMS-processible data definition or schema, usually expressed in a data definition language. If there are physical parameters to be selected in a data definition language, selection of appropriate characteristics are deferred until the physical step. The program design is related to the development of structured programs using the host language and data manipulation language of the DBMS. Conceptual design and implementation design steps are together referred to as logical design by some authors.

4. Physical Design

Physical database design is the process of developing an efficient, implementable physical database structure from a given logical database structure that has been shown to satisfy user information requirements.

Physical database structure represents stored record format, access method, and device allocations for a multiple-record-type database.

Major decision classes of physical design are :

1. Stored record format design. This contains all forms of data representation and compression in stored records. It also contains record partitioning. Record partitioning defines an allocation of individual data items to separate physical devices of the same or different type, or separate extents on the same

device, so that the total cost of accessing data for a given set of user applications is minimized.

2. Access method design. An access method provides storage and retreival capabilities for data stored on physical devices, usually secondary storage. Storage structure and search mechanisms are two important compcnents of an access method. Storage structure defines the limits of possible access paths through indexes and stored records, and the search mechanisms define which paths are to be selected for given applications. A given file may have many associated access paths. Physical databases may require several primary access paths. Efficiency consideration of the dominant application describes the design of individual files. Access time can be greatly reduced through secondary indexes, but at the expense of increased storage space overhead and index maintenance.

3. Stored record clustering. The physical allocatiCn of stored records to physical extents is one of the most important design decisions. Record clustering involves the allocation of records of different types into physical clusters to take advantage of physical sequentiality whenever possible. Analysis of record clustering must take access path configurations into account to avoid access-time degradation due to a new placement of records. Clustering also involves block-size selection for efficient retreival. Blocks in a given clustered extent are influenced by stored record-size and storage characteristics of the physical devices.

IV. DATABASE MODELS

A. INTRODUCTION

A data model is a representation of data and their relationships which describes ideas about the real world. Data models have been used to represent a conceptual view and an implementation view of data. Therefore, we will classify the data models as follows:

1. Conceptual data models
   - Semantic Data Model (SDM)
   - Entity-Relationship (E-R) model
2. Implementation data models
   - Relational data model
   - Hierarchical data model
   - Network data model

One of the major responsibilities of the database administrator is to develop a conceptual model of the organization. The conceptual model is a communications tool between the various users of data, and it is developed without any concern for physical representation.

The conceptual model should be independent of a database management system. The conceptual model has to be mapped to the implementation model used as the underlying structure for a DBMS. The commercial DBMSs available today are based either on a relational data model, hierarchical data model, a network data model, or a combination of them. It is important to understand that the DBMS is not a factor in designing a conceptual model, but designing an implementation model is dependent on the DBMS to be used.

In reality, the DBMS is frequently given, and the database administrator has no choice. The reason for this

situation is that a particular computer may support only one or two DBMS'. On contrast, the choice of the DBMS should be made after the conceptual model is designed. The process of mapping from the conceptual model to the implementation model should be examined while evaluating different DBMS packages. At that time, the DBMS should be a dominant factor when selecting the computer.

E. CONCEPTUAL DATA MODELS

1. Semantic Data Model (SDM)

Contemporary DBMSs are based on database models which have limited capabilities for expressing the meaning of a database. These database models do not adequately relate a database to its corresponding application environment. Therefore, a database model is needed which allows us to capture much more of the meaning of a database. The semantic database model is a higher-level database model and it is designed to provide features for the natural modeling of database application environments. The semantic data model provides a precise documentation and communication medium for database users. More details of the semantic data model will be given in Chapter V.

2. The Entity-Relationship (E-R) Model

The entity-relationship model is a conceptual data model and is based on the view that the real world consists of entities and relationships between entities. In this model, real world objects and their characteristics are represented by entities and their attributes.

An entity is a "thing" which may be distinctly identified; examples are records of officers, units, and so forth. Individual entities are classified into entity sets-that is, collections of entities that may be described by

the same set of properties. Entities with the same attributes fall into one entity set. All OFFICER records form the officer entity set; all UNIT records form the unit entity set. A relationship set is an association between two or more entity sets. The relationship has its own data (e.g., date of assignment, order number of assignment).

Entities and relationships can be represented diagrammatically by an entity-relationship (E-R) diagram. Each entity set is represented by a rectangular box, and each relationship set by a diamond-shaped box in this diagram. The diamond-shaped boxes (relationship sets) are joined to the rectangular boxes (entity sets of entities which participate in the relationship). Figure 4.1 presents a diagram that shows the relationship of the officer and unit entity sets.

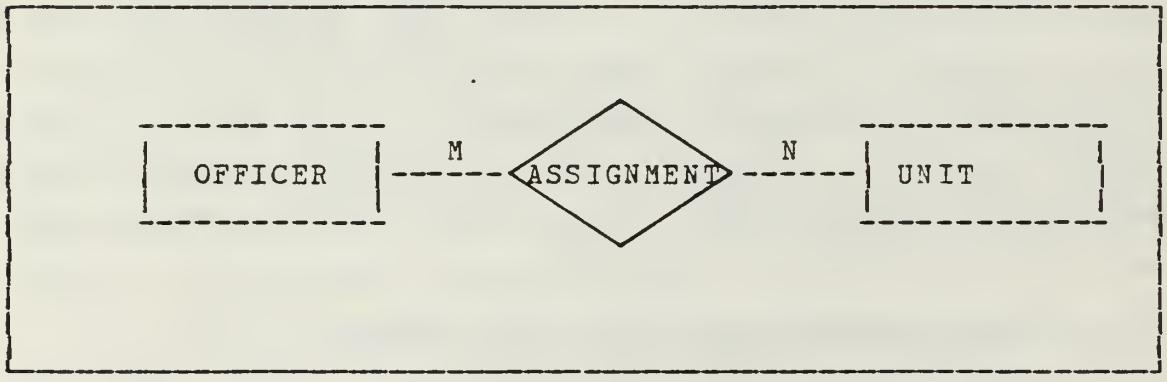


Figure 4.1 E-R Diagram for OFFICER/UNIT Relationship.

OFFICER MID	OFFICER NAME	OTHER DATA
10000	SMITH	
11009	DAVIS	
20112	OZIN	
31052	BUYUKONER	
35278	SARI	
38935	OZKAN	

a. OFFICER Data

UNIT ID	UNIT NAME	OTHER DATA
57BRI	57th BRI	
85DIV	85th DIV	
07REG	7th REG	
35BT	35th BT	
03CC	3rd CO	

b. UNIT Data

OFFICER MID	UNIT ID	DATE OF ASSIGNMENT	ASSIGNMENT ORDER NO
11009	57BRI	781104	104578-9
10000	35BT	790109	125779-1
31052	03CO	790112	363479-6
31052	35BT	801220	563480-7
20112	03CO	810211	258281-6
10000	85DIV	830818	745683-3
32578	35BT	840830	563484-1

c. Relationship Data

Figure 4.2 Three Tables of Data for the E-R Diagram.

C. IMPLEMENTATION DATA MODELS

Implementation data models are chosen to provide constructs that can model a variety of user problems. Most commercial database management systems support a single data model. It is common to classify these models into three classes:

- The relational model
- The hierarchical model
- The network model

The main difference between the three classes of data models lies in the representation of the relationships between the entities.

1. The Relational Data Model

In a relational data model, the entities and their relationships are represented with two-dimensional tables. Every table represents a relation and is made up of rows and columns. Rows of such tables are generally referred to as tuples. Likewise, columns are usually referred to as attributes. Figure 4.3 shows three relations, one for officers, one for units, and the other for assignments.

OFFICER MID	OFFICER NAME	OTHER ATTRIBUTES
10000	SMITH	
11009	DAVIS	
20112	OZIN	
31052	BUYUKONER	
35278	SARI	
38935	OZKAN	

a. OFFICER Relation

UNIT ID	UNIT NAME	OTHER ATTRIBUTES
57BRI	57th BRI	
85DIV	85th DIV	
07REG	7th REG	
35BT	35th BT	
03CO	3rd CO	

b. UNIT Relation

OFFICER MID	UNIT ID	DATE OF ASSIGNMENT	ASSIGNMENT ORDER NO
11009	57BRI	781104	104578-9
10000	35BT	790109	125779-1
31052	03CO	790112	363479-6
31052	35BT	801220	563480-7
20112	03CO	810211	258281-6
10000	85DIV	830818	745683-3
32578	35BT	840830	563484-1

c. ASSIGNMENT Relation

Figure 4.3 Sample Data in Relational Form.

The relational data model approach is a high-level data retrieval and manipulation tool that separates the user from the complexity of storage structures, data structures, and access paths. Access paths do not have to be predefined. The lack of predefined physical access paths means that relational databases must be exhaustively searched to satisfy a querry. The advantages of a relational data model is its simplicity and its well-developed theoretical foundation.

There are several commercially available DBMS packages based on the relational model. Some of them are: IBM's SQL/DS, and System R, Relational Software Inc.'s ORACLE, Relational Technology Inc.'s INGRES, Britton-Lee Inc.'s IDM500, Honeywell's MRDS/LINUS, Ashton-Tate's dbase II, and National Computer Sharing Services' NOMAD. The relational data model will be discussed in greater detail in Chapter VI.

2. Hierarchical Data Model

The hierarchical data model is made up of a hierarchy of the entity types involving a dominant (root) entity type and one or more subordinate (dependent) entity types at the lower levels. The relationship between a dominant and a subordinate entity type is one-to-many. That is, for a given dominant entity there may be many subordinate entity types, and for a given dominant entity occurrence, there can be many occurrences of a subordinate entity type.

The relative simplicity and ease of use the hierarchical data model and the familiarity of data processing users with a hierarchy are major advantages of a hierarchical data model. Disadvantages of a hierarchical data model are:

- The operations of insertion and deletion are complex.
- Any subordinate node is accessible only through its dominant node.

Some examples of commercially available DBMS packages based on hierarchical data model are IBM's IMS, Intel's SYSTEM 2000, and Informatics MARK IV.

3. Network Data Model

The concept of the network data model is based on the work of the CODASYL DBTG (Conference On Data Systems Languages Database Task Group). The network data model employs the set construct. The term set has a different meaning than its mathematical sense.

The network model of a system is diagrammatically represented by a data structure diagram, which was introduced by C.W. Bachman. In this diagram a rectangle enclosing a name denotes an entity or record type. Each record type is composed of data items; but the particular item names are not shown in this description, although they are defined in the complete database description by the data definition language. In a data structure diagram, a directed arrow connects two record types. The record type located at the tail of the arrow is called the owner-record type, and the record type located at the head is called the member-record type. The arrow directed from owner to member is called a set type and it is given a name. Thus the data structure diagram in Figure 4.4 represents the set type ASSIGNED-TO. Here UNIT is the owner record type, and OFFICER is the member record type.

The existence of a set type specifies that there are associations between records of heterogeneous types in the database. This allows the designer to interrelate diverse record types and thus to model associations between diverse entities in the real world.

There is a distinction between a set type and a set occurrence as well as between record type and record occurrence. For example, SMITH and DAVIS denote two record occurrences within the record type OFFICER.

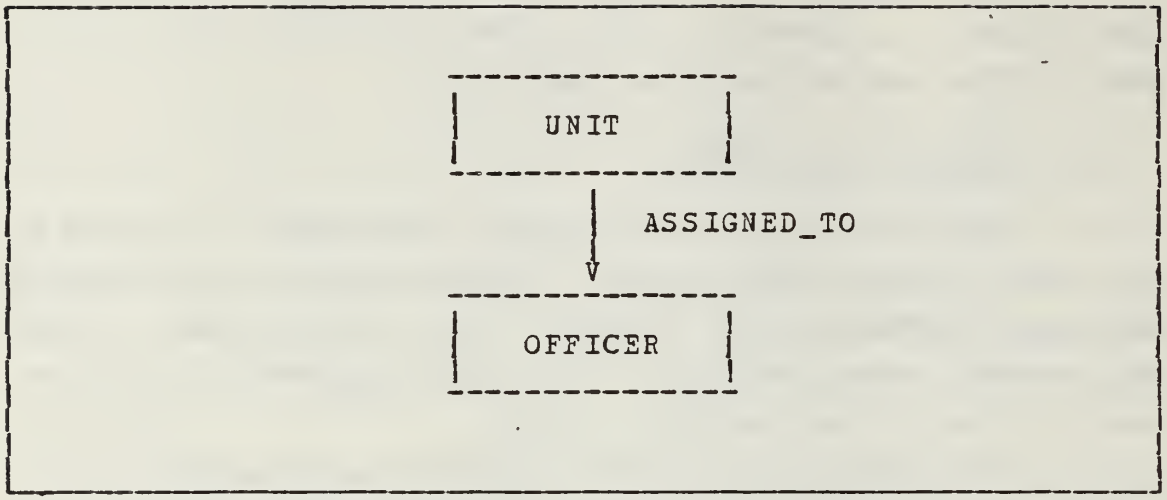


Figure 4.4 A Network Structure.

The existance of a set type is declared by naming it, stating its owner-record type (only one) and its member-record type. A set occurrence is one occurrence of the owner-record type together with zero or more occurrences of each member-record type. This means that there is an occurrence of a set type whenever there is an occurrence of its owner-record type. A set occurrence is an one-to-many relationship that is the basic building block for relating diverse records. The following associations exist among the owner and member records of any set occurrence:

- Given an owner record, it is possible to process the related member records of that set occurrence.
- Given a member record, it is possible to process the related owner record of that set occurrence.
- Given a member record, it is possible to process other member records in the same set occurrence.

Any implementation that conforms to these three rules is a valid implementation of the concept of a set type. Two occurrences of the ASSIGNED-TO set type are shown in Figure 4.5

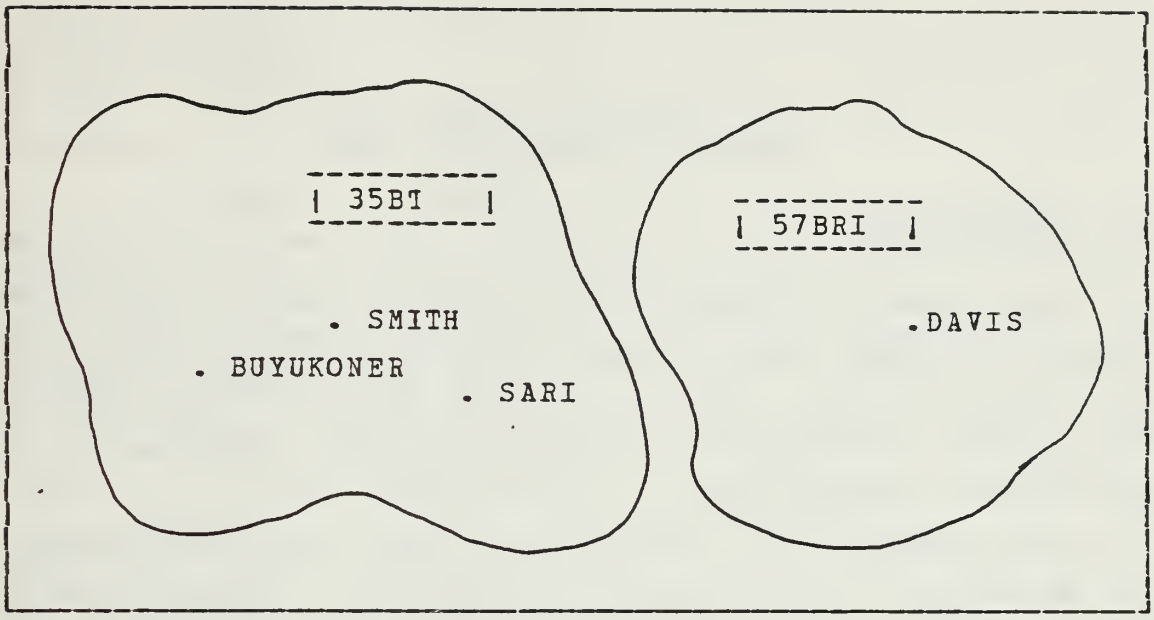


Figure 4.5 Occurences of Set Type ASSIGNED-TO.

A set occurrence with no member-record occurrences is called an "empty set." A given member record may exist in only one set occurrence of a given type. A member record cannot simultaneously belong to two owner records for the same set type.

It is also possible to implement hierarchies (one-to-many relationships) and many-to-many relationships with set structures in the network data model.

The major advantage of the network model is that there are successful database management systems that use the network data model as the basic structure. The main disadvantage of the network model is its complexity.

There are several commercially available database management system packages based on this model. Some of them are: Burroughs' DMS II, CDC's DMS-170, Cullinane's IDMS, Cincom's TOTAL, Honeywell's IDS/II, Univac's DMS1100, Digital Equipment Corporation's DBMS-10/20.

V. SEMANTIC DATA MODEL (SDM)

A. INTRODUCTION

The semantic database model (SDM) was introduced by Hammer and McLeod [Ref. 10:pp. 351-376], and can serve as a conceptual database model in the database design process. The semantic database model allows the same information to be viewed in several ways.

Each database is a model of some real world environment. The contents of a database are intended to reflect a snapshot of the state of this real world environment, and every change to the database should reflect an event occuring in that environment. Therefore, a logical database represents selected portions of reality. Eventually, we may ask questions like: How do we represent the real world environment? What are the structures of the real world environment? Also, we may ask the questions about the other aspect of the problem, such as: How do we represent the conceptual world? What are the structures of the conceptual world?

1. Structures of Real World Environment

The first structure is the object. The real world has objects; they are phenomena that can be represented by nouns. An officer, a unit, an assignment_request are all objects. Objects are grouped into object classes by performing generalization. Objects are grouped together on the basis of similarities. OFFICER is an example of an object class.

Objects have properties. A property is a characteristic of an object. For example, an officer's name and rank are properties. Properties are inherent in objects. The

collection of all possible values of a property is called a property value set. The property value set for officer rank is the collection of all ranks for all possible officer objects.

A fact is an assertion that, for a given object, a particular property has a particular item from the property value set. The statement that the rank of DAVIS is 'captain' is a fact. A fact is the intersection of a given object with a given property value set.

Objects can be related to one another. These relations are called associations. Associations may exist between objects of the same class or of different classes. The association 'commander' exists between objects of the same class (OFFICER). The association 'assignment' exists between two different classes (between OFFICER and UNIT object classes.) Also, an object may have an association to itself. Associations may have properties just as objects have properties. The 'assignment' association may have a property such as Date_of_assignment.

A summary of real world structures is shown in Figure 5.1.

2. Structures of Conceptual World

Database designers should define a conceptual structure for each of the real world structures.

An entity is a conceptual representation of an object. Entities may be grouped into entity classes. An entity class is a representation of an object class. An entity class consists of all the entities that represent the objects of an object class. If there is an object class called OFFICER, then there can be an entity class called OFFICER.

Structure	Definition and Examples
Object	Phenomena that can be represented by nouns. An officer, a unit.
Object Classes	A group of objects formed by generalization. OFFICER, UNIT.
Properties	Characteristics of objects. Name, Rank.
Property value set	The collection of all possible values of a given property. All ranks for Officers.
Fact	The intersection of a given object with a given property value set. Rank of officer DAVIS is captain.
Association	A connection of objects of the same or different classes. DAVIS is assigned to unit ALPHA.

Figure 5.1 Structures in the Real World.

Entities have attributes that are representations of properties of objects. Attributes describe and characterize entities. Rank,Name,Date_of_assignment are examples of attributes.

The conceptual structure that represents property value sets is called a domain. A domain is the collection of all values that an attribute can have. The domain of Names is a collection of character strings of some appropriate length. The domain of height (in centimeters) is the integers from 0 to 250.

A value is the representation of a fact. The value is the intersection of a given entity with a given domain. A relationship is the conceptual representation of an association. Relationships may exist among entities in the same class or in different classes. An entity may have a relationship to itself. A relationship may have attributes, just

as associations may have properties. The assignment relation may have a property such as Date_of _assignment.

Associations between real world structures and conceptual structures are shown in Figure 5.2

Real World Structure	Conceptual Structure
Object	Entity
Object Class	Entity Class
Property	Attribute
Property value set	Domain
Fact	Value
Association	Relationship

Figure 5.2 Real World and Conceptual Structures.

B. GENERAL PRINCIPLES OF DESIGNING SDM

As described in [Ref. 10:p. 355], there are general principles of database organization to support the design of SDM. These are:

"(1)- A database is to be viewed as a collection of entities that correspond to the actual objects in the application environment.

(2)- The entities in a database are organized into classes that are meaningful collections of entities.

(3)- The classes of a database are not in general independent, but rather are logically related by means of interclass connections.

(4)- Database entities and classes have attributes that describe their characteristics and relate them to other database entities. An attribute value may be derived from other values in the database.

(5)- There are several primitive ways of defining interclass connecticns and derived attributes, corresponding to the most common types of information redundancy appearing in database applications. These facilities integrate multiple ways of viewing the same basic information, and provide building blocks for describing complex attributes and interclass relaticnships."

C. DEFINING ENTITY CIASSES

The kasic format of an SDM entity class descriptiсn is shown in Figure 5.3 [Ref. 6:p. 213].

```
ENTITY_CLASS_NAME
  [description: --------------------]

  [interclass ccnnection: ---------------]

    member attributes:

       Attribute_name
         [description: ------------------]
          value class: -----------
         [mandatcry]
         [multivalued][no overlap in values]
         [exhausts value class]
         [not changeable]
         [inverse: Attribute_name]
         [match: Attribute_name1 of ENTITY_CLASS
                   on Attribute_name2]
         [derivation: -------------]

  [class attributes:
       Attribute_name
         [description: ------------------]
          value class: -----------
         [derivation: ------------] ]

  [identifiers:

       Attribute_name1+[Attribute_name2+[...]]]
```

Figure 5.3 Format of SDM Entity Class Descriptions.

An SDM database is a collection of entities. Entities are organized into classes. The structure and organization of an SDM database is specified by an SDM schema. SDM schema identifies the classes in the database. Appendix A is an example of an SDM schema for 'Personnel Database.'

Each entity class in an SDM schema has the following features:

1. A class name identifies the class. Each class name must be unique with respect to all class names used in a schema. OFFICER, UNIT, ASSIGNMENT_REQUEST are all class names.
2. The class has a collection of members (the entities). Each class in an SDM schema is a homogeneous collection of one type of entity.
3. A textual class description is an optional feature of entity class. It describes the meaning and contents of the class.
4. The class has several attributes which describe the members of that class or the class as a whole. There are two types of attributes: Member attributes and Class attributes. For example, each member of class UNIT has attributes Name, Unit_category, Location which identify the unit's name, its category, and its location, respectively. A class attribute describes a property of a class taken as a whole. For example, the class ASSIGNMENT_REQUEST has the attribute Number_of_requests, which identifies the number of requests issued in the current year.
5. An SDM class can either be base or nonbase. A base class is one that is defined independently of all other classes in the database. In Appendix_A the class OFFICER is a base class. It exists independently of other classes. If we think of an entity

class such as COMMANDERS, it is a subset of the OFFICER class, and so can be derived from this class. The class, COMMANDERS is called as an nonbase class, and it does not have independent existance. Every nonbase class has an entry, interclass connection, that describes how the class is to be constructed.

6. If the class is a base class, it has identifiers. These are attributes that uniquely identify members. For example, class OFFICER has the unique identifier, Military_ID.

D. DEFINING ATTRIBUTES

There is a collection of attributes in each class description. These attributes represent the properties of objects. Each attribute has the following features.

1. An attribute name identifies the attribute. Attribute names must be unique within the class where they are defined. They must be unique within all classes that are derived from their class of definition. Date_of_promotion, Main_branch are examples of attribute names.
2. The attribute has a value which is either an entity in the database (a member of some class) or a collection of such entities. The value of an attribute is selected from its underlying value class. Value class is another term for domain that contains the permissible values of the attribute. The value class of an attribute may be any class in the schema or may be the special value NUIL. (i.e., no value.) DATE, BRANCHES are examples of value classes.
3. The applicability of the attribute is specified by indicating that the attribute is either:

(a) a member attribute, which applies to each member of the class, and so has a value for each member (e.g., Military_ID of OFFICER), or

(b) A class attribute, which applies to a class as a whole, and has only one value for the class (e.g., Number_of_requests of ASSIGNMENT_REQUEST.)

4. A textual attribute description is an optional feature that describes tha meaning and purpose of the attribute. This serves as an integrated form of database documentation.
5. The attribute is specified as either single valued or multivalued. A single-valued attribute has one value, that is, a member of the value class of the attribute. The value of a multivalued attribute is a subclass of the value class (e.g., Foreign_language_capability of OFFICER is a multivalued attribute.) The default value for this feature is single valued.
6. An attribute can be specified as mandatory, which means that a null value is not allowed for it. For example, attribute Military_ID of OFFICER is specified as "mandatory"; this models the fact that every OFFICER has a Military_ID.
7. An attribute can be specified as not changeable, which means that once set to a nonnull value, this value cannot be altered except to correct an error. For example, attribute Military_ID of OFFICER is specified as "not changeable."
8. A member attribute can be required to be exhaustive of its value class. This means that every member of the value class of the attribute is used.
9. Finally, multivalued attributes can be specified as nonoverlapping. This means that a member of the value class can be used at most once.

E. MEMBER ATTRIBUTE INTERRELATIONSHIPS

The semantic data model provides three facilities for defining interrelationships among member attributes. These facilities are inversion, matching, and derivation.

1. Inversion

The inverse facility causes two entities to be contained within each other. Member attribute A1 of class C1 can be specified as the inverse of member attribute A2 of C2 which means that the value of A1 for a member M1 of C1 consists of those members of C2 whose value of A2 is M1.

Inverses are always specified by a pair of attributes which establishes a binary association between the members of the classes. For example, in Appendix_A the entity classes OFFICER and UNIT are inverses of each other. In OFFICER, the attribute Unit_assigned has the value class UNIT, and the inverse attribute Officer_assigned. In UNIT, the attribute Officer_assigned has the value class OFFICER, and the inverse attribute Unit_assigned.

2. Matching

The second SDM facility for representing relationships is matching. With matching, a member of one entity class is matched with a member of another entity class. The value of the match attribute A1 for the member M1 of class C1 is determined as follows.

1. A member M2 of some class C2 is found that has M1 as its value of member attribute A2.
2. The value of member attribute A3 for M2 is used as the value of A1 for M1.

For a multivalued attribute (call it A1), it is permissible for each member of C1 to match to several members of C2; in this case, the collection of A3 values is

the value of attribute A1. In other words, the value of an attribute in one of the members is moved to the other.

Inversion and matching provide multiple ways of viewing n-ary associations among entities. Matching supports binary and higher degree associations, while inversion allows the specification of binary associations.

For example, a matching specification in Appendix A indicates that the value of the attribute Foreign_language_capability of a member O of class OFFICER is equal to the value of attribute Foreign_language of the member F of class FOREIGN_LANGUAGE whose FID value is O.

3. Derivation

SDM provides the ability to define an attribute whose value is calculated from other information in the database. Such an attribute is named derived.

The approach is to provide a small vocabulary of high-level attribute derivation primitives that directly model the most common types of derived information. Each of these primitives provides a way of specifying one method of computing a derived attribute.

F. CLASS ATTRIBUTE INTERRELATIONSHIPS

Attribute derivation primitives for member attributes can be used to define derived class attributes, as these primitives derive attribute values from those of other attributes. Of course, instead of deriving the value of a member attribute from the value of other member attributes, the class attribute primitives will derive the value of a class attribute from the value of other class attributes. Moreover, there are two other primitives that can be used in the definition of derived class attributes:

1. An attribute can be defined so that its value equals the number of members in the class it modifies. For example, attribute Number_of_requests is derived from ASSIGNMENT_REQUEST record by summation of members as specified.
2. An attribute can be defined whose value is a function of a numeric member attribute of a class; the functions available are "maximum", "minimum", "average", and "sum" taken over a member attribute. The computation of the function is made over the members of the class.

VI. RELATIONAL DATABASE MODEL

A. RELATIONAL DATA STRUCTURE

In order to explain the relational data structure, it will be very helpful to use the sample data in relational form. Figure 6.1 reflects a relational view of the data which is organized into three tables: OFFICER (officers who are in the army), COURSES (all courses which are offered), and COURSE_ATTENDED (officers who took some courses). The OFFICER table contains, for each officer, a military identification number, rank, name, and the city where the officer was born; the COURSES table contains, for each course, a course code, course name, brief description of that course, duration, and location where the course is offered; and the COURSE_ATTENDED table contains, for each grade, a military identification number, a course code, and a grade taken. The following assumptions regarding officers, courses, and course attended are made. Each officer has a unique military ID number, exactly one rank, name, and city name. Each course has a unique course code, exactly one course name, description of the course, duration, and location. At any given time, no more than one grade exists for a given officer/course combination.

1. Definition of a Relation

Assume that we are given a collection of sets E1, E2, ... ,En (they are not necessarily distinct), R is a relation on those n sets if it is a set of ordered n-tuples <e1, e2, ... , en> such that e1 belongs to E1, e2 belongs to E2, ... , en belongs to En. Sets E1, E2, ... , En are the domains of R. The value n is the degree of R. It is

OFFICER

MID	RANK	NAME	CITY
ID1	Maj	Smith	Berkeley
ID2	Capt	James	Newyork
ID3	Lt	Richard	Monterey

(a)

COURSE_ATTENDED

MID	CCODE	GRADE
ID1	C1	B+
ID1	C2	A
ID1	C3	A-
ID2	C1	A-
ID2	C2	B
ID2	C3	A-
ID2	C4	C+
ID3	C1	A-

(c)

COURSES

CCODE	TITLE	CDESCRIPT	DUR.	LOCATION
C1	Adp	Auto. Data Proc.	16	In.polis
C2	Cobol	Cobol Prog. Lang.	8	Monterey
C3	Digelect	Digital Machines	12	Berkeley
C4	Wepsys	Weapon Systems	114	Monterey

(b)

Figure 6.1 Sample Data in Relational Form.

sometimes called arity n [Ref. 3:pp. 83-93], [Ref. 9:pp. 14-25].

From the mathematical set-theory perspective, we can give another equivalent definition of a relation that is sometimes useful. A relation is any subset of the Cartesian product of one or more domains. For example, if we have n sets, say n=2, E1={a,b}, and E2={0,1,2}, then E1 x E2 is the Cartesian product of these n sets. That is, it is the set of all possible ordered n-tuples <e1,e2> such that e1 belongs to E1, e2 belongs to E2. The result of this Cartesian

product of E1 x E2 is {(a,0),(a,1),(a,2),(b,0),(b,1),(b,2)}. Figure 6.2 , for example, shows the Cartesian product of two sets MID and CCODE (Military ID No., and Course Code).

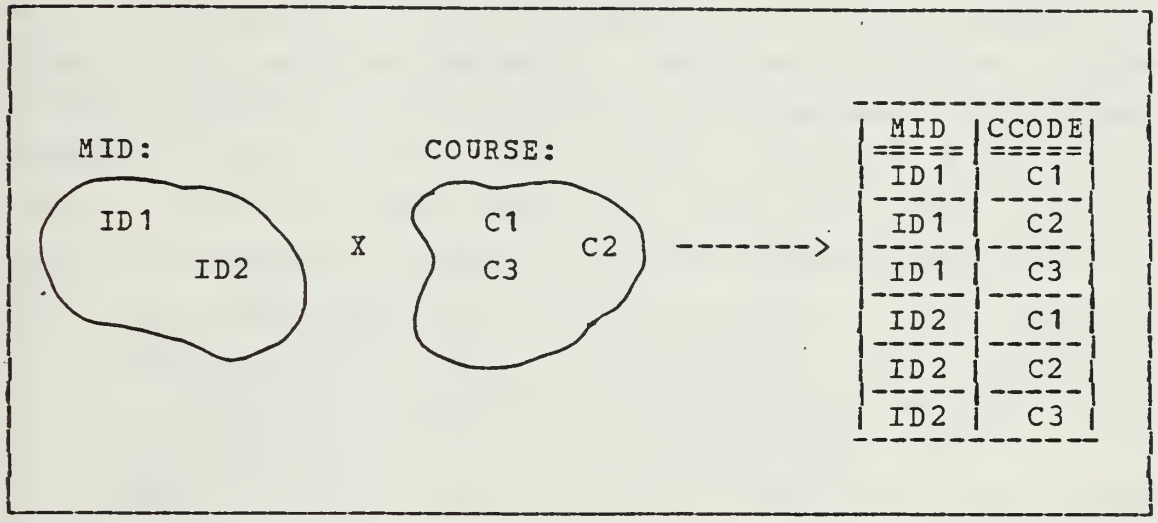


Figure 6.2 An Example of a Cartesian Product.

A relation called COURSES, of degree 5 is illustrated in Figure 6.1 (b). The five domains are sets of values representing, respectively, course codes (CCCDE), course titles (TITLE), brief description of each course (CDESCRIPT), duration for each course, and locations where courses are offered. The "course title" domain, for example, is the set of all valid course titles; note that there may be some titles included in this domain that do not actually appear in the COURSES relation at this particular moment.

It is convenient to view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called attributes. The number of tuples in a relation is called the cardinality of that relation; e.g., the cardinality of the COURSES relation is four, and it has five attributes (or columns). As mentioned earlier, a domain can be thought as a

pool cf values from which the actual values for a given attribute are drawn. It is very important to note that the domains of a relation do have an ordering defined among them. If we have a tuple (a1,a2, ... ,an) with n components, the value of the j th component in this n-tuple has to be drawn from the j th dcmain. In Figure 6.1 (b), (C2, Cobol, Cobol Prog. Language, 8, Monterey) is the second tuple of the CCURSES relation, and the value of the fourth ccmpcnent of this tuple under the attribute named DURATION is drawn from the fourth domain which is a set of positive integers, ranging from 0 to 999. Mathematically speaking, the rearrangement of the five columns of the COURSES relation into some different order results a different relation.

It is important to note the difference between a domain and attributes which are drawn from that domain. An attribute represents the use of a domain within a relation. Figure 6.3 shows a part of a relational schema in which four domains (MILITARY_ID, MILITARY_RANK, OFFICER_NAME, and LOCATICN) and one relation (OFFICER) have been defined by using a data definitiCn language [Ref. 3:pp. 83-93]. The relation is declared with four attributes (MID, RANK, NAME, and CITY), and each attribute is specified as being drawn frcm a corresponding domain. It is sometimes possible that the dcmains of more than one attribute can be the same. In cther words, those attributes can use the same domain in common. To differentiate between attributes that have the same dcmain, each is given a unique attribute name. A crucial feature of relational data structure is that associations between tuples are represented solely by data values in attributes (columns) drawn from a common domain.

All relations in a relational database are required to satisfy the following condition.

> "Every value in the relation (i.e., each attribute value in each tuple) is atomic (i.e., nondecomposable so far as the system is ccncerned)." [Ref. 3:p. 86]

That is, at every row-and-column position in the table there always exists precisely one value, never a set of values. But in the case of having "unknown" or "inapplicable" values, null values can be allowed to represent these special values in a relation. This is the idea of normalization. If a relation satisfies the above condition, it is said to be normalized. This idea will be discussed in detail later.

```
DOMAIN MILITARY_ID        CHARACTER(9)
DOMAIN MILITARY_RANK      CHARACTER(4)
DOMAIN OFFICER_NAME       CHARACTER(20)
DOMAIN LOCATION           CHARACTER(15)

RELATION OFFICER
        (MID       : DOMAIN MILITARY_ID,
         RANK      : DOMAIN MILITARY_RANK,
         NAME      : DOMAIN OFFICER_NAME,
         CITY      : DOMAIN LOCATION)
```

Figure 6.3 Domains and Attributes.

The generalized format or notation which is used to represent a relation is called the relation structure. For example, OFFICER (Mid, Rank, Name, City) is the structure of the OFFICER relation. In general, Relation_name (attribute1, attribute2, ... ,attributeN) is the general format to show the structure of a relation. If we add constraints on allowable data values to the relation structure, we then have a relational schema [Ref. 11].

2. Keys

It is frequently the case that within a given relation there is one attribute with values that are unique within the relation and thus can be used to identify the tuples of that relation. Attribute MID of the OFFICER relation, for example, has this property. Each OFFICER tuple contains a distinct MID value, and this value may be used to distinguish that tuple from all others in the relation. MID is called the primary key for OFFICER.

A single attribute may not always be the primary key in a relation. However, the values of more than one attribute together may constitute a unique identifier. Thus, some combination of attributes, when taken together, have the unique identification property. In the relation COURSE_ATTENDED (Fig. 6.1), for example, the combination (MID, CCODE) has this property. The existence of such a combination is guaranteed by the fact that a relation is a set. Since sets do not contain duplicate elements, each tuple of a given relation is unique with respect to that relation, and hence at least the combination of all attributes has the unique identification property. In the above example, the combination (MID, CCODE) is said to be a composite key as well as a primary key for the OFFICER relation.

On the other hand, occasionally we may encounter a relation in which there is more than one attribute combination having the unique identification property and hence more than one candidate key. In such a case we may arbitrarily choose one of the candidates as the primary key for the relation. If a candidate key is not the primary key, it is called an alternate key [Ref. 3:pp. 83-93]. The COURSES relation in Fig. 6.1 (b) is such a relation. Each course has a unique course code and a unique course name (TITLE).

If the designer chooses one of these candidate keys, say CCODE, as the primary key for the relation, TITLE will be an alternate key.

The primary key is a unique identifier for tuples in a relation. Those tuples represent entities in the real world, and the primary key really serves as a unique identifier for those entities. For example, the tuples in the OFFICER relation represent individual officers, and values of the MID attribute actually identify those officers, not just the tuples that represent them. As a result of this interpretation, we can now introduce the following rules

".Integrity Rule 1 (Entity integrity)
No component of a primary key value may be null."
[Ref. 3:p. 89]

According to the definition, all entities must have a unique identification of some kind. That is, they must be distinguishable from each other. Primary keys perform the unique identification function in a relational database. If a primary key value is null in a relation, this implies that there is some entity that does not have a unique identification. In other words, it is not distinguishable from other entities. It is strongly recommended that both wholly and partially null identifiers be prohibited.

Those types of arguments lead us to a second integrity rule. Occasionally one relation includes references to another. Relation COURSE_ATTENDED, for example, includes references to both the OFFICER relation and the COURSES relation, via its MID and CCODE attributes. It is clearly seen that if an occurrence or a tuple of COURSE_ATTENDED contains a value for MID, say ID2, then a tuple for officer ID2 should exist in OFFICER. Otherwise, the COURSE_ATTENDED tuple would refer to an nonexistent officer; and similarly for courses. To make these notions clear, we should understand the notion of primary domain.

"A given domain may optionally be designated as primary if and only if there exists some single-attribute primary key defined on that domain." [Ref. 3:p. 89]

For example, we may designate the domain MILITARY_ID as primary, by extending its definition shown in Fig. 6.3 as follows:

DOMAIN MILITARY_ID CHARACTER(9) PRIMARY

Any relation which contains an attribute that is defined on a primary domain (for example, relation COURSE_ATTENDED) must obey the following rule.

".Integrity Rule 2 (Referential integrity)
Let D be a primary domain, and let R1 be a relation with an attribute A that is defined on D. Then, at any given time, each value of A in R1 must be either (a) null, or (b) equal to V, say, where V is the primary key value of some tuple in some relation R2 (R1 and R2 not necessarily distinct) with primary key defined on D." [Ref. 3:p. 89]

Here, relation R2 must exist because of the definition of primary domain, and if attribute A is the primary key of R1, the rule is trivially satisfied. When an attribute such as A in one relation is a key of another relation, the attribute is called a foreign key. For example, attribute CCODE of relation COURSE_ATTENDED is a foreign key, because its values are values of the primary key of the COURSES relation.

3. Extentions and Intentions

An extention and an intention are actually components of a relation in a relational database.

The set of tuples existing in a given relation at any given instant is known as the extention of that relation. Thus the extention changes with time. That is, it varies depending upon the several operations performed on tuples which are added, deleted, and updated.

The intention of a given relation is the permanent part of the relation. It is independent of time. The intention corresponds to what is declared in the relational schema. Hence, the intention is the combination of the relation structure (sometimes called the naming structure) mentioned earlier and the integrity constraints which can be subdivided into key constraints, referential constraints, and other constraints. Key constraints are constraints implied by the existence of candidate keys. The primary key specification and the alternate keys specifications included by the intension imply a uniqueness constraint (by the definition of candidate key) and a no-nulls constraint (by Integrity Rule 1) respectively. Referential constraints are constraints implied by the existence of foreign keys. A specification of all foreign keys in the relation implies a referential constraint (by Integrity Rule 2). The relations in Figure 6.1 are examples of extentions and they also show the intentional relation (or naming) structure which consists of the relation name plus the names of the attributes. The operational data appearing under those attributes are the extention part of those tables.

B. RELATIONAL ALGEBRA

Relational algebra is a collection of operations on relations. Each operation takes one or more relations as its operand(s) and produces another relation as its result. A := B + C;, for example, is an arithmetic expression in PASCAL Programming Language. B and C are operands known as variables for the addition operator (+). After performing this operation, the result will be assigned into the variable A. Likewise we encounter B and C as two relations and plus sign (+) as union operator in the relational algebra. After this operation is performed, A will be a new relation produced by that operation as its output or result.

The relational algebra basically consists of two groups of operators: the set operators union, difference, intersection, and product; and the special relational operators selection, projection, join, and division. These operations are very important in order to understand the other high-level relational languages such as SQL, QBE which will be discussed later in this Chapter.

1. Set Operators

The traditional set operators are union, difference, intersection, and Cartesian product. The two relations used as operands must be union-compatible for all except Cartesian product. This means that each relation must have the same number of attributes (same degree), and the attributes in corresponding columns must come from the same domain (the names of the attributes need not be the same). [Ref. 3:pp. 203-215], [Ref. 6:pp. 242-282]

-Union

The union of two relations is formed by combining the tuples from one relation with those of a second relation to produce a third. In other words, the union of two relations A and B, A UNION B, is the set of all tuples t belonging to either A or B (or both). Duplicate tuples are eliminated. For example, let A be the set of officer tuples for officers stationed in Monterey, and B the set of officer tuples for officers who took course C2. Then A UNION B is the set of officer tuples for officers who either are stationed in Monterey or took course C2 (or both).

-Difference

The difference of two relations is a third relation containing tuples which occur in the first relation but not in the second. That is, the difference between two (union-compatible) relations A and B, A MINUS B, is the set of all tuples t belonging to A and not to B. For example, let A and

B be the same sets as in the example under "Union". Then A MINUS B is the set of officer tuples for officers who are staticned in Monterey and who did not take course C2.

-Intersection

The intersection of two relations is a third relation containing common tuples. Again, the relations must be union-ccmpatible. Mathematically speaking, the intersection of twc relations A and B, A INTERSECT B, is the set of all tuples t belonging to both A and B. Let A and B again, for example, be as in the example under "Union" above. Then A INTERSECT B is the set of officer tuples for officers who are stationed in Mcnterey and took course C2.

-Cartesian product

The Cartesian product of two relations is the concatenation of every tuple of one relation with every tuple of a second relation. Iet A and B be two relations. Then A TIMES B or A x B is the set of all tuples t such that t is the concatenaticn of a tuple "a" belonging to A and a tuple "b" belonging to B. The concatenation of a tuple a = (a1,...,aM) and a tuple b = (b1,...,bN), in that order, is the tuple t = (a1,...,aM,bM+1,...,bM+N). For example, let A be the set of all officers' military identification numbers, and B the set of all course code numbers. Then A TIMES B is the set of all pcssible military_ID_number/course_code pairs.

2. Special Relaticnal Operations

-Projection

Projection is an operation that selects specified attributes from a given relation. The result of the projection is a new relation having the selected attributes. In other words, the projection operator creates a "vertical" subset of a given relation obtained by selecting specified attributes, in a specified left-to-right order, and then

eliminating duplicate tuples within the attributes selected. Projection can also be used to change the order of attributes in a relation. For example, consider the COURSE_ATTENDED relation in Figure 6.1 (c). The projection of COURSE_ATTENDED on Ccode and Grade attributes, denoted with brackets as COURSE_ATTENDED {Ccode, Grade}, is shown in Figure 6.4 Note that although COURSE_ATTENDED has eight tuples to begin with, the projection COURSE_ATTENDED{Ccode, Grade} has only six. Two tuples were eliminated because the tuple {C1, A-} and {C3, A-} occurred twice (after the projection was done). Another example of reordering the attributes within the OFFICER relation is to write a statement for projection such as OFFICER {City, Name, Rank, Mid}.

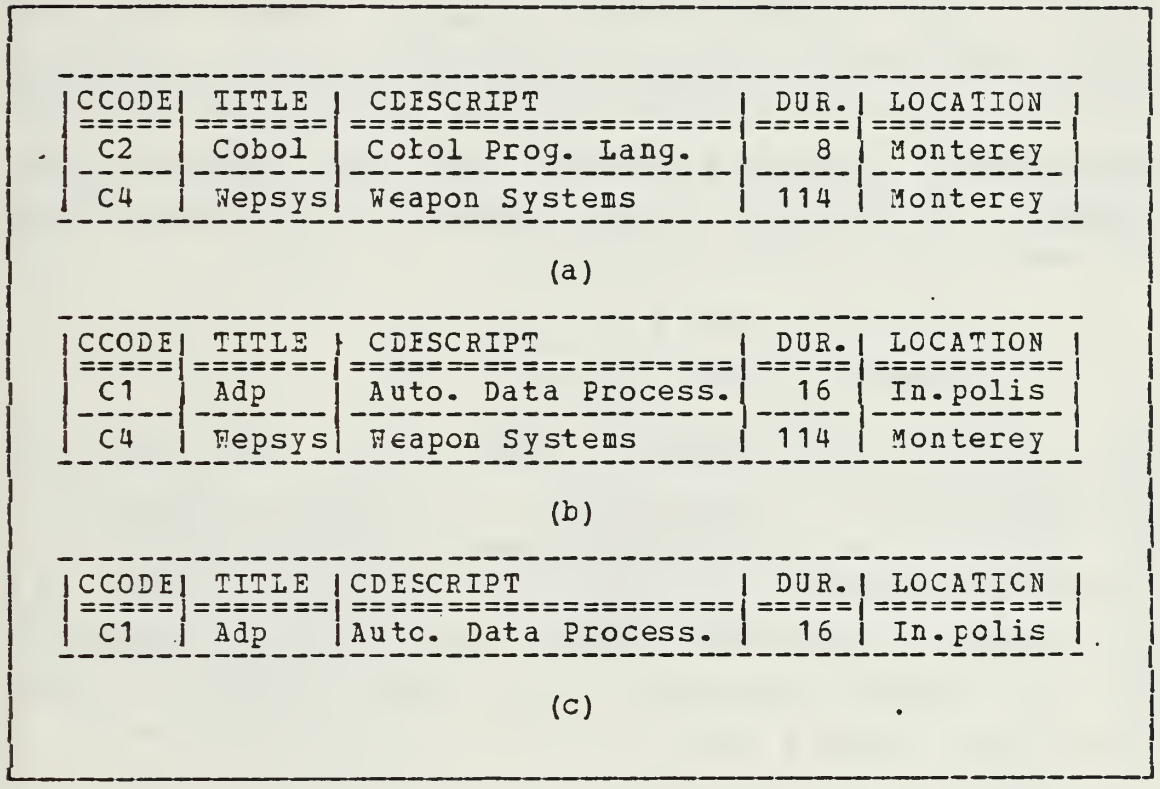
CCODE	GRADE
C1	B+
C2	A
C3	A-
C1	A-
C2	B
C4	C+

Figure 6.4 Projection of COURSE_ATTENDED Relation.

-Selection

The selection operator yields a "horizontal" subset (rows) of a given relation. In other words, selection identifies tuples to be included in the new relation. Selection is denoted by specifying the relation name, followed by the keyword WHERE, followed by a conditional statement involving

attributes. The condition is a single or a combination of Boolean expression(s). Figure 6.5 (a) shows the selection of the relation COURSES WHERE LOCATION = 'MONTEREY'. Figure 6.5 (b) shows the selection of COURSES WHERE DURATION > 12. Figure 6.5 (c) shows the selection of COURSES WHERE DURATION > 12 AND LOCATION = 'INDIANAPOLIS'.



CCODE	TITLE	CDESCRIPT	DUR.	LOCATION
C2	Cobol	Cobol Prog. Lang.	8	Monterey
C4	Wepsys	Weapon Systems	114	Monterey

(a)

CCODE	TITLE	CDESCRIPT	DUR.	LOCATION
C1	Adp	Auto. Data Process.	16	In.polis
C4	Wepsys	Weapon Systems	114	Monterey

(b)

CCODE	TITLE	CDESCRIPT	DUR.	LOCATION
C1	Adp	Auto. Data Process.	16	In.polis

(c)

Figure 6.5 Selection of COURSES Relation.

-Jcin

The join operation is a combination of the product, selection, and (possibly) projection operations. The jcin of two relations, say A and B, is denoted as A JOIN B which is equivalent to taking the Cartesian product of A and B and then performing a suitable selection on that product. If

necessary, duplicate attributes can be eliminated by projection. The Join operation is a binary operation since it operates on two relations but selection and projection are operations on single relations (i.e., they are unary operations).

Actually there are many possible join operations in which the "joining condition" is based on equality or inequality between values in the common column of two relations. Those operations are usually called equijoin, greater-than join, less-than join, and natural join. A natural join is an equijoin with the elimination of duplicate columns, and is a common relational operation. For example, the equijoin and the greater-than join can produce the same result as the expressions

(A TIMES B) WHERE A.X = B.Y
(A TIMES B) WHERE A.X > B.Y

where A and B are relations, and X and Y are attributes belong to A and B, respectively. The values of attributes X and Y must be derived from some common domain. Consider the OFFICER and COURSES relations shown in Figure 6.1 (a) and (b). Tables OFFICER and COURSES may be joined over their CITY and LOCATION attributes; the result is shown in Figure 6.6 We denote such a join as

(OFFICER JOIN COURSES) WHERE OFFICER.CITY = COURSES.LOCATION.

The join in Figure 6.6 is an equijoin. If the duplicate attributes (CITY and LOCATION) were eliminated, then the new relation would be created as a result of the natural join operation.

-Division

The division operation has a binary relation R(X,Y) as the dividend and a divisor that includes Y. The result is

MID	FNK	NAME	CITY	CCODE	TITLE	CDESCRIPT	DUR	LOCATION
ID1	Maj	Smith	Ber.	C3	Digel	Dig.Mach.	12	Berkeley
ID3	Lt	Rich.	Mon.	C2	Cobol	Cob.Prog.	8	Monterey
ID3	Lt	Rich.	Mon.	C2	Cobol	Cob.Prog.	12	Monterey

Figure 6.6 Join of OFFICER and COURSES over CITY and LOC..

a set, S, of values of X such that x belongs to S if there is a tuple (x,y) in R for each y value in the divisor. [Ref. 1:pp. 15-48]

COURSE:

CCODE	LOCATION
C1	Ind.Pol.
C2	Monterey
C3	Berkeley
C4	Monterey
C2	Berkeley

COURSE_LOCATION:

LOCATION
Monterey
Berkeley

CODES:

CCODE
C2

Figure 6.7 The DIVISION Operation.

Figure 6.7 illustrates this operation. If relation COURSE is the dividend and relation COURSE_LOCATION is the divisor, then CODES = COURSE/COURSE_LOCATION. In the Figure, C2 is the only course code for which there is a tuple with Monterey and Berkeley (i.e., <C2, Monterey> and <C2, Berkeley>) in COURSE relation. The other course codes, C1, C3, and C4 do not satisfy this condition.

C. DATA SUBLANGUAGES FOR RELATIONAL DATABASES

The relational database model must provide languages to access relations. A number of relational data sublanguages (DSLs) have been proposed and developed. Because of the tabular structure of relations, users can easily understand relational DSLs. Another important feature of a relational DSL is its selective power. Relational DSLs should have the capability to retrieve data that satisfy any condition over any number of relations. [Ref. 1:p. 36]

Early relational languages were based on selective power. Codd [Ref. 12] gave the definition of the relational model in 1970 and defined the basis for relational languages such as relational algebra and relational calculus. Relational calculus has particular significance. It is a form of predicate calculus specifically tailored to the relational databases and is used to measure the selective power of relational languages. A relational language is relationally complete if it can produce any data that can be obtained from a relational calculus expression. [Ref. 1:p. 36]

Data Sublanguage ALPHA, which is based on relational calculus, was presented by Codd [Ref. 12]. DSL ALPHA itself was never implemented, but a language very similar to it, called QUEL, was used as the query language in the relational DBMS, called INGRES [Ref. 13]. We will discuss INGRES in more detail in Chapter VIII.

Another widely used Data Sublanguage is Structured Query Language (SQL) which is used for and is currently implemented by the System R relational database management system that runs on the IBM System/370 [Ref. 14]. SQL provides retrieval functions and a full range of update operations, and also many other facilities. It can be used both from an on-line terminal and, in the form of "embedded SQL," from an

application program, batch or on-line, written in either COBOL or PL/I. [Ref. 3:pp. 145-156] The basic format, for example, of SQL is in the form

SELECT <attribute>
FROM <relation>
WHERE <conditional expression>

Query-by Example (QBE) is also an another relational system designed for users who are not programmers. It has approximately the same selective power as SQL but uses a graphical interface. It is therefore suitable only for terminal use and cannot be embedded in a host language. [Ref. 1:pp. 181-200] Query-by Example is an artificial, self-contained, user-directed specification language.

So far we have examined several aspects of database systems in general and relational database model in particular. But we have not yet answered the following question: After having a body of data to be represented in a database, how do we decide what relations are needed and what their attributes should be? This is the database design problem which will be discussed in the next Chapter.

VII. RELATIONAL DATABASE DESIGN

A. INTRODUCTION

In designing relational databases, the primary goal is to ensure that relations represent the original data specifications correctly and without redundancy. The major concept for the relational database design is the normalization process, that is, the process of grouping the data elements into relations representing entities and their relationships. The idea of normalization is based on the observation that a certain set of relations has better properties following the database operations, such as inserting, updating, and deleting, than do other sets of relations containing the same data. In other words, the objective of normalization is to produce a database design that can be manipulated in a powerful way with a simple collection of operations while minimizing update anomalies and data inconsistencies [Ref. 15:pp. 99-126]. Normalization theory is a useful aid in the database design process, but it is not an exact solution.

B. NORMAL FORMS

Normalization theory is traditionally expressed through a set of so-called normal forms that progressively constrain the structure and contents of a relation. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints.

There are numerous normal forms which have been defined by the relational theorists. As shown in Figure 7.1 [Ref. 3:p. 239] each of these normal forms contains the other. If a relation, for example, is in third normal form

(3NF), then it is automatically in first and second normal forms. None of these normal forms will eliminate all anomalies; each normal form would eliminate just certain anomalies. But R. Fagin [Ref. 11] defined a new normal form called domain/key normal form (DK/NF), and he showed that a relation in DK/NF is free of all modification anomalies, regardless of their type. The point is to find ways to put relations in DK/NF. If the database designer does this, then he is guaranteed that those relations will have no anomalies. Unfortunately, it is not even known if all relations can be put into DK/NF. At this point we need the concept of functional dependency to define these relational normal forms.

1. Functional Dependency

Functional dependency (FD) is a term derived from mathematical theory; it relates the dependence of values of one attribute or set of attributes on those of another attribute or set of attributes. Formally, an attribute (or set of attributes), Y, in a relation is said to be functionally dependent on another attribute (or set of attributes), X, if knowing the value of X is sufficient to determine the value of Y. To put it another way, there is only one value of Y associated with any value of X. The notation X-->Y is often used to denote that Y is functionally dependent on X, and is read: X functionally determines Y. The attribute (or set of attributes) X is known as the determinant of the FD X-->Y. It is obvious that the nonkey attributes of any relation are functionally dependent on the key.

To illustrate the basic principles of functional dependencies, consider the sample database in Figure 6.1. The attribute TITLE in relation COURSES is functionally dependent on CCODE because each course has one given title value. Thus once a course code is known, a unique value of

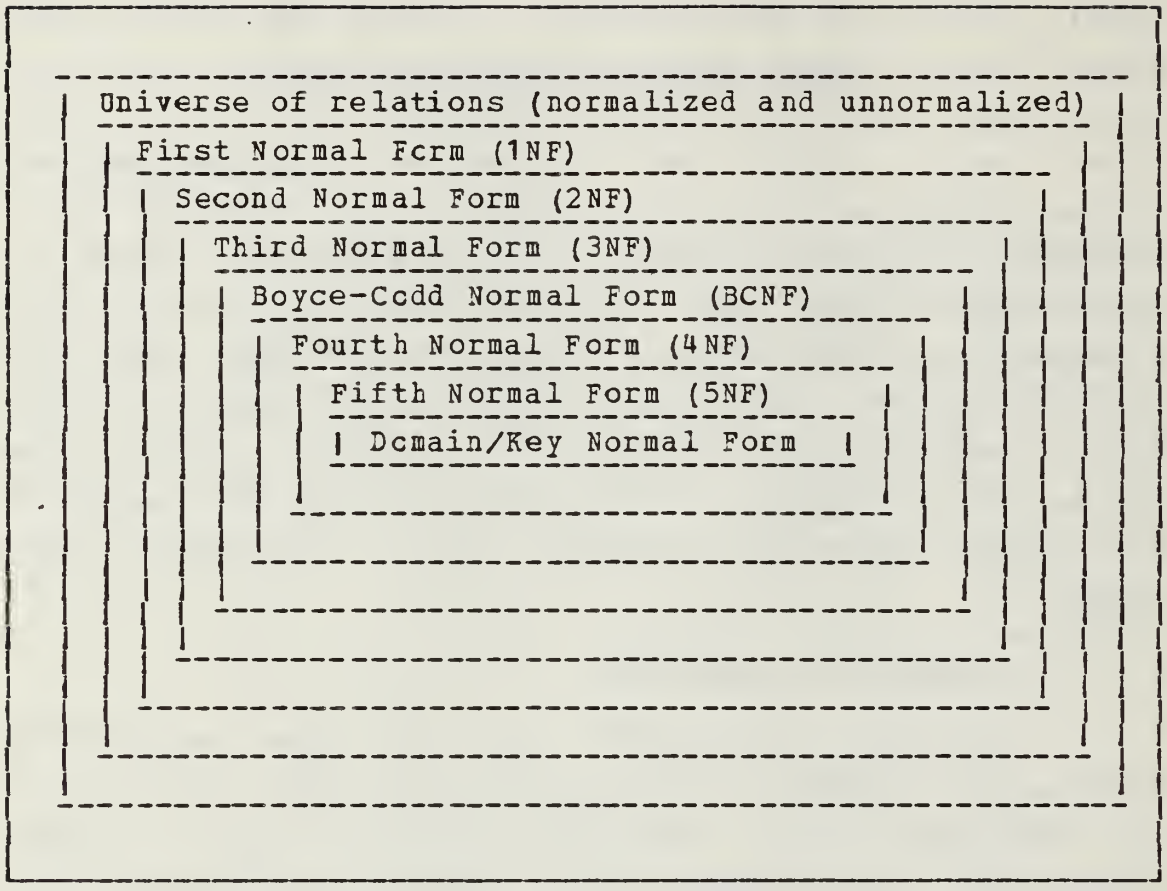


Figure 7.1 Relational Normal Forms.

course title is immediately determined. The FD for this example is shown as CCODE --> TITLE.

Likewise, in relation COURSE_ATTENDED, once values for officer ID (MID) and CCODE are known, a unique value of GRADE for that officer in that course is determined. This FD is defined as MID,CCODE --> GRADE.

It is convenient to represent the FDs in a given set of relations by means of a functional dependency diagram, an example of which is shown in Figure 7.2 It is also possible to have two attributes that are functionally dependent on each other. In this case both CCODE --> TITLE and TITLE --> CCODE hold (because TITLE is an alternate key for the

relation COURSES). The notation CCODE <---> TITLE is commonly used to illustrate such mutual FD.

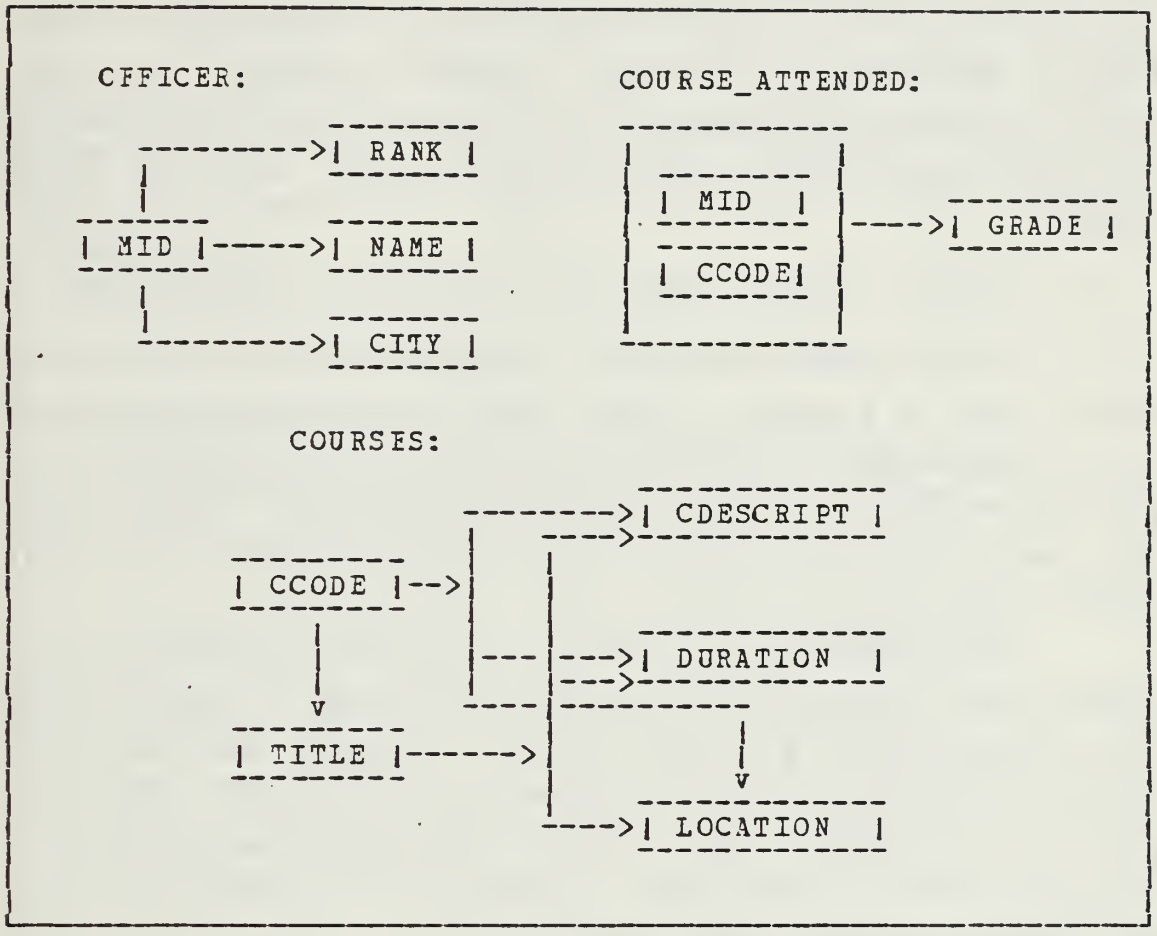


Figure 7.2 Functional Dependency Diagrams.

We also need to introduce the concept of full functional dependency. This term is used to show the minimum set of attributes in a determinant of an FD. [Ref. 1:pp. 15-48] Attribute (or set of attributes) Y is said to be fully functionally dependent on attribute (or set of attributes) X if Y is functionally dependent on X and Y is not functionally dependent on any proper subset of X. For example, in the relation COURSE_ATTENDED, the attribute

GRADE is fully functionally dependent on the attributes (MID, CCODE) because it is not functionally dependent on either MID or CCODE alone. On the other hand, in the relation COURSES, the attribute CDESCRIPT is functionally dependent on the attributes (CCODE, TITLE); however, it is not fully functionally dependent on those attributes because, it is also functionally dependent on either CCODE or TITLE alone.

2. First, Second, Third, and Boyce-Codd Normal Forms

First normal form (1NF) deals with the "shape" of a record type or a tuple. Under first normal form, all tuples in a relation must have the same set of attributes, and the attributes must be atomic (indivisible items). This definition merely states that any normalized relation is in first normal form.

When determining whether a particular relation is in normal form, the FDs between the attributes in the relation must be examined. For this reason, we will use a notation which was first proposed by [Ref. 16] to point out these relational characteristics. In the notation, the relation is defined as divided into two components: the attributes and the FDs between them. The format is

R = ({X,Y,Z},{X-->Y,X-->Z})

R is the name of the relation, X, Y, and Z are the attributes, and X-->Y, X-->Z are FDs. For example, in Figure 6.1 the relation COURSE_ATTENDED is defined as

COURSE_ATTENDED=({MID,CCODE,GRADE},{MID,CCODE-->GRADE})

Many update and deletion anomalies can be eliminated by converting a relation to second normal form (2NF). Second normal form requires that all nonkey attributes must contain information that refers to the entire key, not just part of

it. In other words, a relation is said to be in 2NF if and only if it is in 1NF and every nonkey attribute of the relation is fully functionally dependent on the primary key. The relation UNIT_ASSIGNED, for example, in Figure 7.3 is in 1NF but not in 2NF because the nonkey attribute GSTATUS (geographical status) is not fully dependent on the primary key UCODE (unit code) and MID. Here GSTATUS is fully functionally dependent on UCODE, which is a subset of the primary key.

Relation: UNIT_ASSIGNED Key: UCODE,MID

UCODE	LOCATION	GSTATUS	MID	DATE
U1	Monterey	100	ID1	012583
U1	Monterey	100	ID4	042385
U1	Monterey	100	ID5	012581
U1	Monterey	100	ID2	110182
U2	Newyork	200	ID6	083084
U3	Denver	300	ID3	072882
U3	Denver	300	ID5	100584
U4	Newyork	200	ID1	031584

Figure 7.3 Relation in 1NF but not in 2NF.

In Figure 7.4 relation UNIT_ASSIGNED has been decomposed into two relations, UNITS and ASSIGNMENT. Both relations are in 2NF. Note that the relation UNIT_ASSIGNED suffers from modification anomalies with respect to update operations. Figure 7.5 also illustrates the FDs for both relations.

Problems occur with each of the following three basic operations.

UNITS key: UCODE

UCODE	LOCATION	GSTATUS
U1	Monterey	100
U2	Newyork	200
U3	Denver	300
U4	Newyork	200

ASSIGNMENT key: MID,UCODE

MID	UCODE	DATE
ID1	U1	012583
ID1	U4	031584
ID2	U1	110182
ID3	U3	072882
ID4	U1	042385
ID5	U1	012581
ID5	U3	100584
ID6	U2	083084

Figure 7.4 Relations in 2NF.

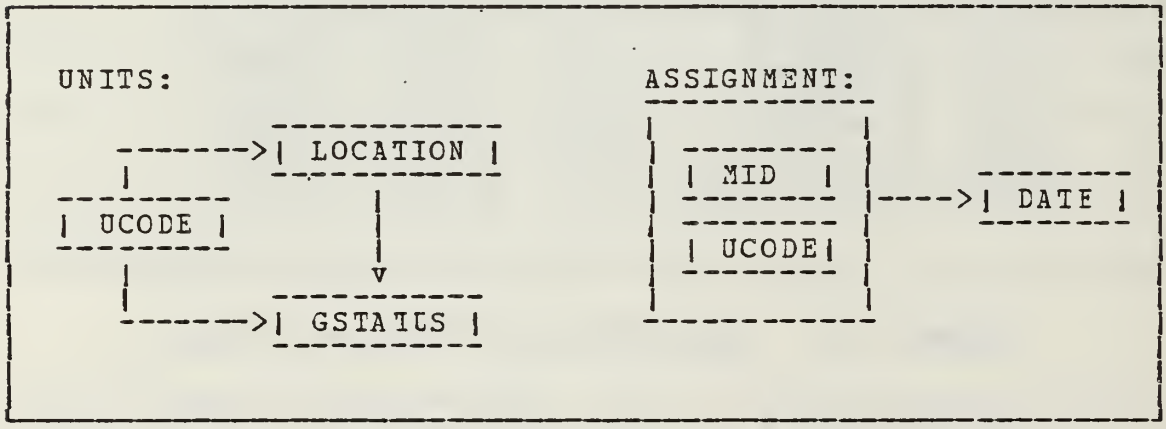


Figure 7.5 FD Diagrams for UNITS and ASSIGNMENT.

Inserting: We cannot enter the fact that a particular unit is located in a particular city until at least one officer is assigned to that unit.

Deleting: If we delete the only tuples for a particular unit, we lose both the assignment dates for officers who have been assigned to that unit and the information that the unit is located in a particular city.

Updating: There is significant redundancy in the relation UNIT_ASSIGNED. This redundancy causes update problems. For example, the values of both attributes LOCATION and GSTATUS appear in UNIT_ASSIGNED many times.

The solution to these problems is to decompose the relation UNIT_ASSIGNED into two relations UNITS (UCODE, LOCATION, GSTATUS) and ASSIGNMENT (MID, UCODE, DATE) shown in Figure 7.5 .

The anomalies remaining in a 2NF relation can be eliminated by converting it to third normal form (3NF). The definition of 3NF is that the relation meets the requirements of 2NF, and, in addition, that each nonkey attribute refers directly to the key [Ref. 15:pp. 99-126]. The other definition of 3NF is given by [Ref. 3:p. 248].

> "A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key."

Relation UNITS in Figure 7.4 is not in 3NF since the attribute GSTATUS is transitively dependent on the primary key UCODE. That is, UCODE ---> LOCATION, and LOCATION ---> GSTATUS. In order to put the relation UNITS into 3NF, it can be decomposed into two relations by doing two projection operations. In this case, relations UC1 (UCODE, LOCATION) and U1G (LOCATION, GSTATUS) are both 3NF. Figure 7.6 shows these relations corresponding to the data values of the original UNITS (Fig. 7.4). A relation that is in 2NF and not in 3NF can always be decomposed into an equivalent collection of 3NF relations [Ref. 3:pp. 237-265].

UCL Key: UCODE

UCODE	LOCATION
U1	Monterey
U2	Newyork
U3	Denver
U4	Newyork

ULG Key: LOCATION

LOCATION	GSTATUS
Denver	300
Monterey	100
Newyork	200

Figure 7.6 Relations in 3NF.

The original definition of 3NF was subsequently replaced by a stronger definition known as Boyce/Codd Normal Form (BCNF) which can be defined as follows.

> "A relation R is in Boyce/Codd Normal Form (BCNF) if and only if every determinant is a candidate key." [Ref. 3:p. 249]

The original 3NF definition does not satisfactorily handle the case of a relation that has more than one candidate key, and modification anomalies arise with this definition when it is used with such relations. BCNF is often used to remove these anomalies. For example, consider the relation UNIT_ASSIGNED (Fig. 7.3) and the FDs between the attributes of that relation such as

```
UNIT_ASSIGNED=({UCODE,LOCATION,GSTATUS,MID,DATE},
{UCODE-->LOCATION, UCODE-->GSTATUS, LOCATION-->GSTATUS,
UCODE,MID-->DATE})
```

Here the relation UNIT_ASSIGNED contains three determinants but only the determinant (UCODE, MID) is a candidate key. Therefore UNIT_ASSIGNED is not BCNF. Similarly, UNITS (Fig. 7.4) is not BCNF, because the determinant LOCATION is not a candidate key. On the other hand, relations ASSIGNMENT, UCL,

and ULG are each BCNF, because in each case the candidate key is the only determinant in the relation.

BCNF is conceptually simpler than 3NF since it does not reference the concepts of primary key, transitive dependency, and full dependency. Although BCNF is stronger than 3NF, it is still true that any relation can be decomposed in a nonloss way into an equivalent collection of BCNF relations.

3. Forth and Fifth Normal Forms

Forth and fifth normal forms deal with multivalued attributes. A multivalued attribute may correspond to a many-to-many relationship, as with officers and skills, or to a many-to-one relationship, as with the children of an officer. By "many-to-many" we mean that an officer may have several skills and/or a skill may belong to several officers. When we look at the many-to-one relationship between children and fathers, it is a single-valued fact about a child but a multivalued fact about a father. In some sense, 4NF and 5NF are also related with composite keys. These normal forms attempt to minimize the number of attributes involved in a composite key. [Ref. 17]

Forth normal form is based on the concept of multivalued dependency (MVD). The notation X-->>Y is used to indicate that a set of attribute Y is multidependent on a set of attributes of X. Formally, MVD is defined as follows: Given a relation R with attributes X, Y, and Z, the multivalued dependency X-->>Y holds in R if and only if the set of Y-values matching a given (X-value, Z-value) pair in R depends only on the X-value and is independent of the Z-value. The attributes X, Y, and Z may be composite. [Ref. 3:pp. 237-265]

Multivalued dependencies which have been defined can exist only if the relation R has at least three attributes.

It is easy to show that, in the relation R(X,Y,Z), the MVD X-->>Y holds if and only if the MVD A-->>C holds.

Before giving the definition of 4NF, it is convenient to state the following theorem proved by Fagin in [Ref. 18].

> "Relation R, with attributes X, Y, and Z, can be nonloss-decomposed into its two projections R1(X,Y) and R2(X,Z) if and only if the MVD X-->>Y,Z holds in R."

Now fourth normal form (4NF) is defined as follows:

> "A relation R is in fourth normal form (4NF) if and only if, whenever there exists an MVD in R, say X-->>Y, then all attributes of R are also functionally dependent on X (i.e., X-->Z for all attributes Z of R)." [Ref. 3:p. 259]

Fagin also proves (see [Ref. 18]) that 4NF is strictly stronger than BCNF (i.e., any 4NF relation is necessarily in BCNF), and any relation can be nonloss-decomposed into an equivalent collection of 4NF relations.

Fifth normal form (5NF) deals with cases where information can be reconstructed from smaller pieces of information which can be maintained with less redundancy. 2NF, 3NF, and 4NF also serve this purpose, but 5NF generalizes to cases not covered by the others. Aho and co-workers in 1979 [Ref. 19] discovered relations that cannot be nonlosslessly decomposed into two relations but can be losslessly decomposed into three or more relations. Because of this property, 5NF is also called projection-join normal form, and is based on the concept of join dependency (JD) which is a more general case of an MVD. In general, relation R satisfies the JD *(X,Y,...,Z) if and only if it is the join of its projections on X,Y,...,Z, where X,Y,...,Z are subsets of the set of attributes of R [Ref. 3:pp. 237-265]. We can now define 5NF given by [Ref. 3:p. 262].

> "A relation R is in fifth normal form (5NF)-also called projection-join normal form (PJ/NF)- if and only if every join dependency in R is implied by the candidate keys of R."

Since a JD is a more general case of an MVD, any relation which is in 5NF is necessarily in 4NF. But determining that a relation is in 5NF is less straight-forward than 4NF, BCNF, etc. because discovering join dependencies is a nontrivial task.

4. Domain / Key Normal Form

In 1981, R. Fagin [Ref. 11] defined a new normal form called domain/key normal form (DK/NF). In his paper he proved that a relation in DK/NF will have no insertion or deletion anomalies. He also showed that a relation having no modification anomalies must be in DK/NF. DK/NF is based on only the concepts of key and domain. These concepts are readily known and supported by DBMS products. The definition of DK/NF is quite simple.

> "A relation is in DK/NF if every constraint on the relation is a logical consequence of the definition of keys and domains." [Ref. 6:p. 299]

In this definition, constraint is a broad term. Any rule on static values of attributes that can be evaluated precisely whether or not it is true is said to be a constraint. Thus FDs, MVDs, JDs, and edit rules are all examples of constraints. Some constraints which have to do with changes in data values are excluded from the definition of constraint.

DK/NF relation requires that if keys and domains can be defined such that all constraints will be satisfied when the key and domain definitions are satisfied, then modification anomalies are impossible. But there is no known way to put a relation in DK/NF automatically. In spite of this problem, DK/NF can be extremely useful for practical database design. DK/NF is a design objective. Database designers wish to define their relations such that constraints are logical consequences of domains and keys. This goal can be accomplished for many designs.

Seven normal forms have been discussed and are summarized in Figure 7.7 [Ref. 6:p. 305].

Form	Defining Characteristic
1NF	Any relation.
2NF	All nonkey attributes are dependent on all of the keys.
3NF	There are no transitive dependencies.
4NF	Every MVD is a functional dependency.
5NF	Join dependencies are satisfied.
DK/NF	All constraints on relations are logical consequences of domains and keys.

Figure 7.7 Summary of Normal Forms.

C. RELATIONAL DESIGN PROCEDURES AND CRITERIA

1. Design Procedures

The relational model is attractive in database design since it provides formal criteria for logical structure, namely, normal form relations. In order to produce those relations, database designers should choose a design procedure. Two different approaches have been proposed:

> "1. Decomposition procedures. These commence with a set of one or more relations and decompose nonnormal relations in this set into normal forms.
>
> 2. Synthesis procedures. These commence with a set of functional dependencies and use them to construct normal form relations." [Ref. 1:p. 59]

In practical situations, synthesis procedures are more attractive than decomposition procedures. Many algorithms have been proposed for relational design and each algorithm

produces relations that satisfy some subset of the relational design criteria which will be discussed in the next Section.

Decomposition algorithms start with one relation and successively decompose it into normal form relations. The relations in 3NF and BCNF are not sufficient for applying these decomposition algorithms, so the ideas of MVD and 4NF have to be known. Synthesis algorithms, on the other hand, start with a set of FDs and synthesize them into normal form relations. In other words, these algorithms use FDs to produce normal form relations. Detail information about design algorithms can be found in [Ref. 1:pp. 59-88].

2. Relational Database Design Criteria

This Section presents several different design criteria which have been identified in [Ref. 6:pp. 307-311] and [Ref. 16] for producing an effective relational database.

a. Elimination of Modification Anomalies

The objective of this criterion is to eliminate all anomalies resulting from database operations. As we have seen, if relations are in DK/NF, then no modification anomalies can occur. This is why DK/NF is a design objective. The problem is to find a way that all relations can be put into DK/NF.

b. Relation Independence

According to this design goal, two relations are said to be independent if modifications can be made to one without regard for the other. However, this criterion is not always achievable. Interrelation constraints allow relations to be dependent. To eliminate this dependency the relations can be joined together. After the join operation, the new relation may have modification anomalies. To eliminate these anomalies, relations are decomposed into two or more

relations; but this cperation creates interrelation dependencies again. Here we see the conflict in design goals. In this case we must chcose the least of the evils, based on the requirements of the application.

c. Ease of Use

This third criterion for a relational database design makes the relations seem natural to users. As far as possible, designers should attempt to structure the relations so that they are familiar to users. From time to time this criterion conflicts with the other two criteria.

d. Representation

This relaticnal criterion states that the final structure has to correctly represent the original specifications. That is, all the relations in the output design process must satisfy the conditions for normal form. C. Beeri and co-workers have defined three important pcints for the representation of a set of relations, Sin, in the input design process by a set of relations, Sout, in the output design process (Sin ard Sout are sets of relations used in the input and output design processess):

"-REP1: The relaticns Sout contain the same attributes as Sin.

-REP2: The relaticns Sout contain the same attributes and the same FDs as Sin.

-REP3: The relaticns Sout contain the same attributes and the same data as Sin." [Ref. 1:p. 63]

The first representation, REP1, requires all the attributes in Sin to also be in the relations in Sout. But it does not address any dependencies between the attributes.

In regard to REP2, representation requires that each FD in Sin be either contained as an FD in one of the relations in Sout or derived from the FDs in the relations in Sout, using the FD inference rules.

The third representation criterion, REP3, also requires that the relations in Sout contain exactly the same tuples as the original relations in Sin.

e. Separation

The separation criterion means that the original specifications are separated into relations that satisfy certain conditions. As we have discussed earlier in this Chapter, the database must be divided into a number of normal form relations.

f. Redundancy

This last criterion points out the fact that the final structure must not contain any redundant information. It is possible to define the redundancy criterion in different ways. One set of redundancy criteria is shown below:

> "-RED1: A relation in Sout is redundant if its attributes are contained in the other relations in Sout.
>
> -RED2: A relation in Sout is redundant if its FDs are the same or can be derived from the FDs in the other relations in Sout.
>
> -RED3: A relation in Sout is redundant if its content can be derived from the contents of other relations in Sout." [Ref. 1:p. 66]

Here, RED1 is not a very useful idea, because during decomposition it is often necessary to create separate relations that represent FDs between attributes, which may appear in other relations. RED2 and RED3, however, can be very useful criteria. Any design algorithms should avoid RED3 because it would keep the same data in more than one relation.

The design criteria discussed in this Section can conflict. When conflicts occur, the designer has to assess priorities and make the best possible compromise in light of requirements. There is no single rule of priority.

D. TRANSFORMING THE SDM INTO RELATIONAL MODEL

Figure 7.8 illustrates the logical design of the Personnel database (see SDM in Chapter V). This logical schema cannot be used to implement the relational personnel database for the following reasons:

1. Most of the relations have multivalued attributes. Such attributes cannot be used in a relation,
2. The logical schema allows some tuples to be contained in other tuples. The relations must be normalized or redefined to eliminate these inconsistencies.

As shown in Figure 7.8, inversion, matching, and derivation have been used to provide interrelationships between the attributes. Inverse and match functions must be eliminated in order to achieve DK/NF. During this process, the new interrelation constraints should be added.

Initially, the relationships between OFFICER and ACADEMIC_EDUCATION, MILITARY_EDUCATION, MEDICAL_INFO, and FOREIGN_LANGUAGE were assumed as one-to-many. For example, an officer can have more than one medical report, and many reports may belong to one officer. Such relationships were described by match function in the SDM design. On the other hand, relations OFFICER and UNIT have many-to-many relationships with each other, and these relationships were defined by the inverse function in the same SDM. The relationship, for example, between unit_assigned and officer_assigned is many-to-many. To eliminate this problem, a new relation called ASSIGNMENT has been constructed.

By considering all those conditions, rules, design criteria, etc. described in this Chapter, the resulting relational design (relational schema) is illustrated in Figure 7.9, and domain definitions with attribute/domain correspondences are shown in Figure 7.10 and Figure 7.11 respectively. For simplicity, some attributes are removed in

```
OFFICER (Military_ID, Rank, Date_of_promotion, Name,
         Birth_date, Beginning_date_to_active_duty,
         Native_country, Sex, Marital_status, Number_
         of_children, Permanent_address, Current_
         address, Primary_branch, Secondary_branch,
         Academic_education, Military_education,
         Health_condition,Foreign_language_capability,
         Unit_assigned)

        Key: Military_ID
  Notes: 1. Academic_education is a contained
            ACADEMIC_MAJOR tuple, multivalued.
         2. Military_education is a contained
            MILITARY_EDUCATION/COURSES tuple,
            multivalued.
         3. Health_condition is a contained ME-
            DICAL_INFO tuple, multivalued.
         4. Foreign_language_capability is a
            contained FOREIGN_LANGUAGE tuple,
            multivalued.
         5. Unit_assigned is a contained UNIT
            tuple, multivalued.

UNIT (Unit_code, Name, Unit_category, Location,
      Superior_unit, Unit_function, Officer_assigned)

        Key: Unit_code
  Note:  Officer_assigned is a contained OFFICER
         tuple, multivalued.

ACADEMIC_MAJOR (Academic_branch, Academic_degree, AID,
                Date, Name_of_university)

        Key: (Academic_branch,Academic_degree,AID)

MILITARY_EDUCATION/COURSES (Course/Military_school_
        code, Location, MEID, Course/School_title,
        Duration, Date, Grade)

        Key: (Course/Military_school_code, Location,
              MEID)

MEDICAL_INFO(Medical_report_number, HID, Date, Height,
        Weight, Blood_pressure, Eye_condition, Ear_
        condition, Internal, General_health_status)

        Key: (Medical_report_number, HID)

FOREIGN_LANGUAGE (Name_of_language, FID, Degree_of_
                  capability)

        Key: (Name_of_language, FID)

ASSIGNMENT_REQUEST (Unit_code, Request_number, Date,
        Primary_branch_requested, Secondary_branch_
        requested, Academic_major_requested, Military/
        course_education_requested, Medical_status,
        Number_of_person, Number_of_requests)

        Key: (Unit_code, Request_number)
```

Figure 7.8 Summary of Logical Design.

the relational schema in Fig. 7.9, and they are referred to the attribute OTHERS. A sample of the designed database with example data is given in Appendix B.

In order to be familiar with some currently available DBMSs, the INGRES IBMS will be introduced in Chapter VIII and the sample personnel database in Appendix B will also be implemented by using another DBMS known as ORACLE in Chapter IX.

```
OFFICER (MID,Rank,Name,Sex,Pri_bran,Sec_bran,Others)
     Key: MID
UNIT (Ucode,Uname,Ucat,Uloc,Sup_unit,Ufunc)
     Key: Ucode
A_EDUCATION (Abran,Adeg,AID,Univ,Gdate)
     Key: (Abran,Adeg,AID)
M_EDUCATION (Ccode,Cloc,MEID,Cgrade,Cdate)
     Key: (Ccode,Cloc,MEID)
M_COURSES (Ccode,Cloc,Ctitle,Cdesc,Cdur)
     Key: (Ccode,Cloc)
MEDICAL (Repno,HID,Rdate,Eyecond,Earcond,Hstat,Others)
     Key: (Repno,HID)
LANGUAGE (Nlanguage,FID,Ldegree)
     Key: Nlanguage
ASGREQ (R_ucode,Reqnum,Reqdate,R_rank,R_pribr,
        R_secbr,R_acabr,R_miled,R_hstat,Numofpers)
     Key: (R_ucode,Reqnum)
ASSIGNMENT (AMID,A_ucode,Orderno,Asgdate)
     Key: (AMID,A_ucode,Asgdate)
```

Figure 7.9 Relational Schema for Personnel Database.

DOMAIN NAME	FORMAT and MEANING
MID	numeric 999999999
RANK	CHAR(4); abbreviations of military ranks for the army
PERSON_NAMES	CHAR(20); names of officers
SEX	CHAR(1); value is 'M' or 'F'
BRANCHES	CHAR(8); abbreviations of military branches for the army
UNIT_CODE	CHAR(6); unit codes
UNIT_NAMES	CHAR(15); names of units
UNIT_CAT	CHAR(4); unit categories ('div','hosp', etc.)
LOCATION	CHAR(15); names of locations
UNIT_FUNC	CHAR(6); unit functions
ACADEMIC_BRANCHES	CHAR(5); abbreviations of academic branches
ACADEMIC_DEGREES	CHAR(3); value is 'BA','BS','DR', 'MA','MS', or 'ENG'
UNIVERSITY_NAMES	CHAR(10); names of universities
DATE	CHAR(9); format is DD-MMM-YY
COURSE/SCH_CODE	CHAR(6); course or school codes
COURSE/SCH_TITLES	CHAR(10); titles of courses
COURSE/SCH_DESC.	CHAR(30); description of courses
COURSE_GRADES	CHAR(2); value is 'A','A-','B+', 'B','B-','C+','C','C-', 'D','F','P', or 'X'
EYE_EAR_CONDITION	numeric 99; codes for eyes and ears
HEALTH_STATUS	numeric 99
LANGUAGES	CHAR(10); names of foreign languages
LANGUAGE_CAP.	numeric 9
ORDER_NO	CHAR(8); format is '999999-9'
OTHERS	subclass of STRINGS where specified
INTEGERS	numeric values where specified

Figure 7.10 Domain Definitions.

ATTRIBUTE	DOMAIN
AID,AMID,FID, HID,MID,MEID	MID
RANK,R_RANK	RANK
CNAME	PERSON_NAMES
SEX	SEX
PBI_BRAN,R_PRIBR, SEC_BRAN,R_SECBR	BRANCHES
A_UCODE,R_UCODE, UCODE,SUP_UNIT	UNIT_CODE
UNAME	UNIT_NAMES
UCAT	UNIT_CAT
ULOC,CLOCB	LOCATION
UFUNC	UNIT_FUNC
ABRAN,R_ACABR	ACADEMIC_BRANCHES
ADEG	ACADEMIC_DEGREES
UNIV	UNIVERSITY_NAMES
ASGDATE,CDATE,GDATE, RDATE,REQDATE	DATE
CCODEA,CCODEB, E_MILED	COURSE/SCH_CODE
CTITLE	COURSE/SCH_TITLES
CDESC	COURSE/SCH_DESCRIPTION
CGRADE	COURSE_GRADES
CDUR,NUMOFPERS	INTEGERS
CRDERNO,REPNO, REQNUM	ORDER_NO
EYECOND,EARCOND	EYE_EAR_CONDITION
HSTAT,RHSTAT	HEALTH_STATUS
NLANGUAGE	LANGUAGES
LDEGREE	LANGUAGE_CAPABILITY
OTHERS	OTHERS

Figure 7.11 Domains and Attributes for Personnel Database.

VIII. INGRES - A RELATIONAL DATABASE SYSTEM

A. INTRODUCTION

INGRES(Interactive Graphics and Retrieval System) is a relational database and graphics system which is implemented on top of the UNIX operating system developed at Bell Telephone Laboratories. The implementation of INGRES is primarily programmed in "C", a high level language in which UNIX itself is written. Parsing is done by using YACC, a compiler-compiler available on UNIX.

INGRES runs as a normal user job on top of the UNIX operating system. The primary significant modification to UNIX that INGRES requires is a substantial increase in the maximum file size allowed.

In this chapter we shall describe some of the principal components of INGRES. These include the query language QUEL, INGRES utility commands, and the storage structures supported.

B. QUEL: A RELATIONAL QUERY LANGUAGE

QUEL (QUEry Language) is a calculus based language. Each interaction of QUEL contains one or more range-statements of the form

RANGE OF variable_list IS relation_name

The purpose of this statement is to specify the relation over which each variable ranges. The variable_list portion of a RANGE statement declares variables which will be used as arguments for tuples. These are called tuple variables.

Each QUEL interaction also includes one or more state-ments of the form

Command [result_name](target_list)
[WHERE Qualification]

Here command is either RETRIEVE,APPEND, REPLACE, or DELETE. We use square brackets ([ ]) to denote "zero or more". For RETRIEVE and APPEND, result_name is the name of the relation which qualifying tuples will be retrieved into or appended to. For REPLACE and DELETE, result_name is the name of a tuple variable which identifies tuples to be modified or deleted. The target_list is a list of the form

result_domain = QUEL Function ...

Here the result_domains are domain names in the result relation which are to be assigned the values of corresponding functions.

The goal of a query is to create a new relation for each RETRIEVE statement. The relation so created is named by the "result_name" clause and the domains in that relation are named by the "result_domain" names given in the target_list. The result_domain name may be omitted and is then taken to be the same as the Domain_name in the function. The result-name is an optional parameter to designate that the table returned by the query be permanently stored in the database with the result_name as its identifier. Retrievals that specify a result_name do not display the result table on the terminal screen. The result_name cannot be the name of an existing table.

To create the desired relation, first consider the product of the ranges of all variables which appear in the target_list and the qualification of the RETREIVE statement. Each term in the target_list is a function and the Qualification is a truth function, i.e.; a function with values true or false, on the product space. The desired relation is created by evaluating the target_list on the

subset of the product space for which the Qualification is true, and eliminating duplicate tuples.

The QUEL examples in this chapter all concern the following relations.

```
OFFICER(MID,RANK,CITY)
COURSE(CCODE,TITLE,CDESCRIPT,DUR,LOCATION)
COURSE_ATTENDED(MID,CCODE,GRADE)
```

The following are valid QUEL interactions.

Example 1. Compute duration multiplied by 7 for course Wepsys.

```
RANGE OF C IS CCURSES
RETRIEVE INTO W
(DUR_IN_DAYS = C.DUR * 7)
WHERE C.TITLE = "Wepsys"
```

Here C is a tuple variable which ranges over the COURSES relation, and all tuples in that relation are found which satisfy the qualification C.TITLE = "Wepsys". The result of the query is a new relation, W, which has a single domain DUR_IN_DAYS that has been calculated for each qualifying tuple. If the resulting relation is omitted, qualifying tuples are written in display format on the user's terminal or returned to a calling program.

Example 2. Insert the tuple (ID4,Capt,John,Salinas) into OFFICER relation.

```
APPEND TO OFFICER(MID = "ID4",RANK = "Capt",
     NAME = "John",CITY = "Salinas")
```

Here the resulting relation OFFICER is modified by adding the indicated tuple to the relation. Domains which are not specified default to zero for numeric domains and null for character strings.

Example 3. Cancel all the courses which are given in Monterey.

```
RANGE OF C IS COURSES
DELETE C WHERE C.LOCATION = "Monterey"
```

Here C specifies that the COURSES relation is to be modified. All tuples are to be removed for which LOCATION has the value "Monterey".

Example 4. Promote all captains to major if the officer got the grade 'A' from any course.

```
RANGE OF O IS OFFICER
RANGE OF CA IS COURSE_ATTENDED
REPLACE O(RANK = "Maj")
WHERE O.RANK = "Capt" AND
      O.MID = CA.MID AND CA.GRADE = "A"
```

Here O.RANK is to be replaced by "Maj" for those tuples in OFFICER relation where the qualification is true.

C. INGRES UTILITY COMMANDS

In addition to the above QUEL commands, INGRES supports a variety of utility commands. These utility commands can be classified into seven major categories.

1. Invocation of INGRES:

   INGRES database_name

   This command invokes INGRES. "Database_name" which is the name of an existing database. (A database is simply a named collection of relations with a given database administrator.) This command executed from UNIX "logs in" a user, then the user may issue all other commands (except those executed directly from UNIX) within the environment of the invoked database.

2. Creation and destruction of databases:

CREATEDB database_name

DESTROYDB database_name

These two commands are called from UNIX. The CREATEDB command creates a new INGRES database. The person who executes this command becomes the Database Administrator (DBA) for the database. DESTROYDB command removes all references to an existing database. The directory of the database and all files in that directory are removed. To execute DESTROYDB that person must be the DBA for "database_name".

3. Creation and destruction of relations:

CREATE tablename (columnname = format,columnname = format,...)

DESTROY tablename

The CREATE command enters a new table into the database. The table is "owned" by the user who invokes the command. DESTROY removes the table from the database. Only the table owner may destroy a table. The columns are created with the type specified by "format". The current formats accepted by INGRES are 1-, 2-, and 4-byte integers, 4- and 8-byte floating point numbers, and 1- to 255-byte fixed length ASCII character strings.

4. Bulk copy of data:

COPY tablename(columnname = format,columnname = format,...) into|from "filename"

PRINT tablename

The COPY command moves data between INGRES tables and standard files. "Tablename" is the name of an existing table. In general, "columnname" identifies a column in the table. "Format" indicates the storage format for the column's values in the file. To write a file, use the "into filename" form of the COPY command. To copy data from a file to an INGRES table, use the "from filename" form of the command. The PRINT command displays the contents of a table specified at a user's terminal under predefined formats.

.5. Storage structure modification:

```
MODIFY tablename TO storage_structure
                    ON(key1,key2,... )

INDEX ON tablename IS indexname(key1,key2,... )
```

The MODIFY command changes the storage structure of a relation from one access method to another. Only the owner of a table can modify that table. This command is used to accelerate performance of queries that access the table, particularly when the table is large or frequently referenced. The storage structures currently supported will be discussed in Section D of this chapter. The indicated keys are domains in tablename which are used concatenated left to right to form a combined key which is used in the organization of tuples in all but one of the access methods. The INDEX command creates a secondary index on existing tables in order to make retrieval and updating with secondary keys more efficient. The secondary key is constructed of columns from the primary table in the order given. A maximum of six "columnname"s may be specified per index, but a user can build any number of secondary indexes for a

primary table. Only the owner of a table is allowed to create secondary indexes on that table. In order to maintain the integrity of the index, users are not permitted to update secondary indexes directly. However, wherever a primary table is changed, its secondary indexes are automatically updated by the system.

6. Consistency and integrity control:

DEFINE INTEGRITY ON range_var IS qual

DESTROY INTEGRITY tablename(integer,...,integer|all)

HELP INTEGRITY tablename

RESTORE database_name

The DEFINE INTEGRITY command adds an integrity constraint for the table referred to by "range_var". After the constraint is defined, all updates to the table must satisfy "qual". "Qual" must be true for every existing row in the table when the INTEGRITY statement is issued. Updates that violate any integrity constraints are simply not performed.

HELP INTEGRITY command prints current integrity constraints on a specified table. DESTROY INTEGRITY removes integrity constraints from a table. To destroy constraints for a table, the integer arguments should be those printed by a HELP INTEGRITY command on the same table. Only the table owner may destroy integrity constraints.

The RESTORE command checks and cleans up a database after an INGRES or operating system crash. RESTORE should be executed after any abnormal termination to assure database integrity. The RESTORE command is only available to the database administrator.

7. Miscellaneous:

HELP

SAVE tablename UNTIL expiration_date

PURGE database_name

HELP may be used to print information about INGRES features, definitions of views, protections or permissions, or information about the contents of the database and specific tables in the database. SAVE is the mechanism by which a user can declare his intention to keep a table until a specified time. PURGE is a UNIX command which can be invoked by a database administrator to delete all relations whose "expiration_dates" have passed. This should be done when space in a database is exhausted. (The database administrator can also remove any relations from his database using the DESTROY command, regardless of who their owners are.)

D. STORAGE STRUCTURES

Often the relation (table) will be stored in such a way that a complete scan is not required. Also secondary indices can be declared and are used if possible to limit the number of tuples examined.

There are five modes of relation storage structure. A relation owner can decide both storage structure and what secondary indices (if any) to construct, then both decisions will be done automatically by the system. The five main storage structures are:

1. ISAM : indexed sequential access method structure, duplicate rows removed
2. CISAM : compressed isam, duplicate rows removed

3. HASHED : random hash storage structure, duplicate rows removed
4. CHASH : compressed hash, duplicate rows removed
5. HEAP : unkeyed and unstructured

For the first four structures the key may be any ordered collection of domains. These schemes allow rapid access to specific portions of a relation when key values are supplied. The remaining non_keyed scheme (a "HEAP") stores tuples in the file independently of their values and provides a low overhead storage structure, especially attractive in situations requiring a complete scan of the relation.

IX. IMPLEMENTATION OF PERSONNEL DATABASE USING ORACLE DBMS

The CRACLE relational DBMS has been used to implement the Personnel Database because of its simplicity and clarity. SQL is the language that is used to access and control data in an ORACLE database. As a result of this, SQL is used as DSL in the database operations such as table and view creation, updating data, and in queries. There are nine relations in the Personnel Database. A sample of the designed database with example data is shown in Appendix B. A relation can be created using CREATE command. An example of CREATE command to create OFFICER's relation can be as follows:

```
UFI> CREATE TABLE OFFICER
  2     ( MID          NUMBER(5) NOT NULL,
  3       RANK         CHAR(4),
  4       ONAME        CHAR(9),
  5       SEX          CHAR(1),
  6       PRI-BRAN     CHAR(6),
  7       SEC-BRAN     CHAR(6),
  8       OTHERS       CHAR(8) );

Table created.
```

After the relation is created, tuples of OFFICER can be inserted using the INSERT command.

```
UFI> INSERT INTO OFFICER
  2  VALUES(27363,'capt','Johnson','m','artill','pilot');

1 record created.

UFI> INSERT INTO OFFICER
  2  VALUES(12239,'maj','Hernandez','m','inftry','spefc');

1 record created.
```

Users of the Perscnnel Database wish to get scme information by asking the following sample queries.

1. List all tuples in the relation OFFICER using SELECT command.

```
UFI> SELECT *
  2  FROM OFFICER;

    MID RANK ONAME      SEX PRI-BRAN SEC-BRAN OTHERS
------- ---- --------- --- -------- -------- --------
  27363 capt Johnson   m   artill   pilot
  12239 maj  Hernandez m   inftry   spefc
  32458 1lt  Roobins   f   airdef   adp
  43596 2lt  Smith     m   medic    pilot
  10999 lcol Brown     m   inftry   adp
  35768 1lt  Greenberg m   sigcor   pilot
  29364 capt James     m   milenq   spefc
  16745 maj  Leighton  m   financ   adp
  10792 col  Stone     m   ordnan   artill

9 records selected.
```

2. List all officers who were assigned between the date 1-JAN-1966 and 1-JAN-1980.

```
UFI> SELECT MID,RANK,ONAME
  2  FROM OFFICER
  3  WHERE MID IN
  4      ( SELECT AMID
  5        FROM ASSIGNMENT
  6        WHERE ASGDATE BETWEEN '1-JAN-66' AND '1-JAN-80');

    MID RANK ONAME
------- ---- ---------
  10792 col  Stone
  10999 lcol Brown
  16745 maj  Leighton
  27363 capt Johnson
  29364 capt James
  12239 maj  Hernandez

6 records selected.
```

3. List military_IDs, ranks, names, and primary branches of all officers who have taken an 'adp' course.

```
UFI> SELECT MID,RANK,ONAME,PRI←BRAN
  2  FROM OFFICER
  3  WHERE MID IN
  4       ( SELECT MEID
  5         FROM M←EDUCATION
  6         WHERE CCODEA IN
  7             ( SELECT CCODEB
  8               FROM M←COURSES
  9               WHERE CTITLE = 'adp' ));

     MID RANK ONAME      PRI←BRAN
------- ---- --------- --------
  32458 1lt  Robbins    airdef
  10999 1col Brown      inftry
```

4. List all unit categories for units.

```
UFI> SELECT UNIQUE UCAT
  2  FROM UNIT;

UCAT
----
div
brg
dep
hoso
reg
bt
bn

7 records selected.
```

5. List military_IDs, ranks, names, sex, and primary branches for all officers who speak German and took the course 'SS002'.

```
UFI> SELECT MID,RANK,ONAME,SEX,PRI+BRAN
  2  FROM OFFICER
  3  WHERE MID IN
  4       ( SELECT MEID
  5         FROM M+EDUCATION
  6         WHERE CCODEA = 'SS002' )
  7  AND MID IN
  8       ( SELECT FID
  9         FROM LANGUAGE
 10         WHERE NLANGUAGE = 'german' );

    MID RANK ONAME      SEX PRI+BRAN
------- ---- --------- --- --------
  35768 1lt  Greenberg m   sigcor
```

6. Order all military_education tuples by course_code, and within course_code put them in descending course_grade order.

```
UFI> SELECT *
  2  FROM M+EDUCATION
  3  ORDER BY CCODEA,CGRADE;

CCODEA    MEID CGRADE CDATE
------ ------- ------ ---------
AA102    43596 A-     12-JUL-84
AD002    32458 A      23-NOV-82
AS003    27363 B+     13-APR-77
CS302    32458 B+     26-OCT-84
CS509    10999 A-     31-JAN-77
HS706    43596 A      22-JUN-82
IA076    16745 A-     22-NOV-76
IS005    10999 A      11-OCT-70
IS005    12239 A-     30-SEP-72
OC092    10792 A-     01-DEC-69
SS002    35768 B+     26-FEB-82

11 records selected.
```

A view may derive data from more than one relation. This is done by defining a view using a join query. An example of a view (view OFFICEREDUC) is shown below.

7. Create an OFFICEREDUC view from a join of the OFFICER relation to the A_EDUCATION relation.

```
UFI> CREATE VIEW OFFICEREDUC(VID,RANK,VNAME,VBRAN,VDEG,SEX) AS
  2  SELECT MID,RANK,ONAME,ABRAN,ADEG,SEX
  3  FROM OFFICER,A_EDUCATION
  4  WHERE OFFICER.MID = A_EDUCATION.AID;

View created.
```

8. Count the number of officers who are captain and have 'BS' academic degree.

```
UFI> SELECT COUNT(RANK)
  2  FROM OFFICEREDUC
  3  WHERE VDEG = 'BS'
  4  AND   RANK = 'capt';

 COUNT(RANK)
------------
           2
```

9. List all assignments, ordered by assignment date, for officer identified by '10792' as MID.

```
UFI> SELECT MID,ONAME,UNAME,ASGDATE
  2  FROM OFFICER,ASSIGNMENT,UNIT
  3  WHERE MID = 10792 AND MID = AMID
  4  AND UCODE = A←UCODE
  5  ORDER BY ASGDATE;

    MID ONAME      UNAME           ASGDATE
------- --------- --------------- ---------
  10792 Stone     9th Art Brg     01-SEP-66
  10792 Stone     2nd Inf Div     11-JAN-71
  10792 Stone     64th Ord Depot  15-APR-78
```

10. List all officers and courses for officers in terms of the courses with duration of at least one year.

```
FI> SELECT RANK,ONAME,CDESC,CLOCB,CDUR
 2  FROM OFFICER,M←COURSES,M←EDUCATION
 3  WHERE MID = MEID
 4  AND CCODEA = CCODEB
 5  AND CDUR >= 52;

ANK ONAME     CDESC                       CLOCB            CDUR
--- --------- --------------------------- -------------- -------
1t  Smith     Aca. of Health Sci.         Ft.Houston,TX       96
1t  Smith     Army Aviation School        Ft.Rucker,AL        52
```

11. Compute and display the sum of duration of course(s) which was taken by officer 'Smith'.

```
UFI> SELECT SUM(CDUR)
  2  FROM OFFICER,M-COURSES,M-EDUCATION
  3  WHERE MID = MEID
  4  AND ONAME = 'Smith'
  5  AND CCODEA = CCODEB;

 SUM(CDUR)
----------
       148
```

12. List all officers and assignment requests, including rank, officer name, military_ID, and request number, for assignment requests where officers' specialties meet the requirements specified in that request.

```
UFI> SELECT REQNUM,MID,RANK,ONAME
  2  FROM OFFICER,ASGREQ,A-EDUCATION,M-EDUCATION,MEDICAL
  3  WHERE MID = MEID
  4  AND MID = AID
  5  AND MID = HID
  6  AND PRI-BRAN = R-PRIBR
  7  AND SEC-BRAN = R-SECBR
  8  AND    ABRAN = R-ACABR
  9  AND CCODEA   = R-MILED
 10  AND HSTAT <= R-HSTAT;

REQNUM       MID RANK ONAME
-------- ------- ---- ---------
327685-4   12239 maj  Hernandez
031084-3   16745 maj  Leighton
```

X. FUNCTIONS OF A DATABASE MANAGEMENT SYSTEM

A. INTRODUCTION

The principal function of the DBMS is to store, retrieve, and modify data. However in an operational environment the DBMS should provide other important functions. In this chapter, we will first describe major DBMS functions and then discuss three of these functions in detail : recovery, concurrency, and security.

Major DBMS functions are described by Codd, E.F. in [Ref. 20:p. 114], and they are shown in Figure 10.1.

1. Store, retrieve, and update data.
2. Provide recovery services in case of failure.
3. Provide concurrency control services.
4. Provide security facilities.
5. Provide integrity services to enforce database constraints.
6. Provide a user-accessible catalog of data descriptions.
7. Support logical transactions.
8. Interface with communications control programs.
9. Provide utility services.

Figure 10.1 Major Functions of a DBMS.

The first and fifth functions have been discussed in previous chapters. Recovery, concurrency control, and security facilities will be discussed later in this chapter.

The management of large, complex databases is difficult. Maintaining a database with ten record types and hundreds of data-items can be time-consuming. If a database is processed by hundreds of application programs then changes of records and data-items can be risky. Questions like "Which programs are affected by a change to eight-digit Unit_Location_Codes?" or "Which records contain MID?" are frequent. Since most databases are self-describing, much of the data needed to answer these questions are stored within the database. However, these information may not be readily accessed by humans. For that reason, "a user-accessible catalog" which contains data descriptions and data about the relationship between programs and data will be very useful to the user, and it should be provided by DBMSs.

A logical transaction is a sequence of activities performed atomically. Usually, transactions include several actions on the database. Unfortunately, the DBMS product cannot know which groups of actions are logically related. Thus the DBMS must provide facilities for the application programmer to define transaction boundaries which are needed in handling concurrent control and recovery functions.

In addition to these functions, the DBMS must interface with a communications control subsystem which controls the flow of transactions to application programs from the DBMS. Finally, the DBMS must provide utility programs to facilitate database maintenance. These utility programs may be used to unload, reload, and execute the database; or they may be used to make mass insertions or deletions of data in or out of the database.

No current DBMS provides all of these functions in a satisfactory way. These capabilities can be used as a checklist of decision criteria for a DBMS. A system that does not provide most of them is not truly a DBMS.

B. RECOVERY

Computer and database systems can fail in many ways. Computers stop unexpectedly, disk heads crash, operators may drop disks, programs may have bugs, and so on. Database systems must include not only a variety of checks and controls to reduce the likelihood of failure, but also an extensive set of procedures for recovering from the failures that will inevitably occur despite those checks and controls.

In an operational environment there are many possible causes of failures, such as:

- programming errors; in an application or in the database system,
- hardware errors; on the device or the channel or the CPU,
- operator errors; such as mounting a wrong tape,
- fluctations in the power supply,
- fire in the computer system room.

If such errors occur during a database interaction, the database can be left in an inconsistent state. Recovery software is used to restore the database to some previous consistent state.

1. Recovery via Reprocessing

There are a variety of recovery algorithms. The simplest way is to keep back-up a copy of a database. This copy is created periodically, once or twice a day. Then, when a failure occurs, the last back-up copy is used to restore the database. Any transactions since that copy was made are run again. This algorithm is called "recovery via reprocessing" and it has several drawbacks. First, reprocessing transactions takes the same amount of time as processing them the first time. This means that one day will

be required to recover one day of processing. If the system is heavily loaded, the system may never catch up. Second, when transactions are processed concurrently, it is impossible to guarantee that they can be reprocessed in the same order as they were originally processed. For these reasons, reprocessing is not a viable form of recovery.

2. Transactions

The fundamental purpose of the database system is to process transactions. A transaction is a program, or a program part, that can read from or write into the database. It consists of the execution of an application-specific sequence of operations. These operations can be of five types: BEGIN TRANSACTION, READ, WRITE, COMMIT, and ROLLBACK. All transactions begin with BEGIN TRANSACTION operation. READ causes a page or record to be read from the database. WRITE causes a new copy of a page or record to be written into the database. COMMIT tells the system that the transaction has terminated succesfully and that all of its updated pages or records should be permanently reflected in the database. ROLLBACK tells the system that the transaction has terminated abnormally and that the records or pages it wrote into should be returned to their previous state. A transaction can have only one COMMIT or ROLLBACK processed, and transactions cannot be nested.

The recovery manager processes the READ, WRITE, COMMIT, and ROLLBACK commands. It also handles system failures so it provides reliability for the DBMS.

3. Recovery via Rollback/Rollforward

This approach uses the following four step algorithm:

1. Recreate the outputs of all successfully completed transactions. (Transactions which are ended with COMMIT operation.)
2. Abort all transactions in process at the time of failure.
3. Remove database changes generated by aborted transactions.
4. Restart aborted transactions.

This algorithm will appropriately recover the database. Some transaction outputs cannot be undone. This outputs are called "Real outputs" by Gray in [Ref. 21:pp. 223-242]. Real outputs are messages which are received by people who are using the system, like order confirmations or inputs to other transactions. The message "OFFICER,(MID:9999999), IS ASSIGNED TO THE UNIT, (UNIT_ID:9999), ASSIGNMENT ORDER NO IS 99999-9" are examples of real outputs. Because they cannot be undone, real outputs should not be produced until the transaction is completed. It is recommended that a log of real outputs be maintained. When the transaction is completed, the actions onthe log are updated and the real outputs become visible. If a failure occurs when the real outputs are being produced, each output could be numbered and a log kept of the real outputs that have been produced.

4. Transaction Logging

To apply UNDO (rolling back a transaction) and REDO (rollforward a transaction) processes to a database system, a log should be kept of transaction results. The log includes the old and new values of all items updated by the transaction, and it is in chronological order. The log resides on either disk or tape. When a failure occurs, the log is used to both UNDO and REDO transactions, as shown in Figure 10.2 and Figure 10.3, respectively.

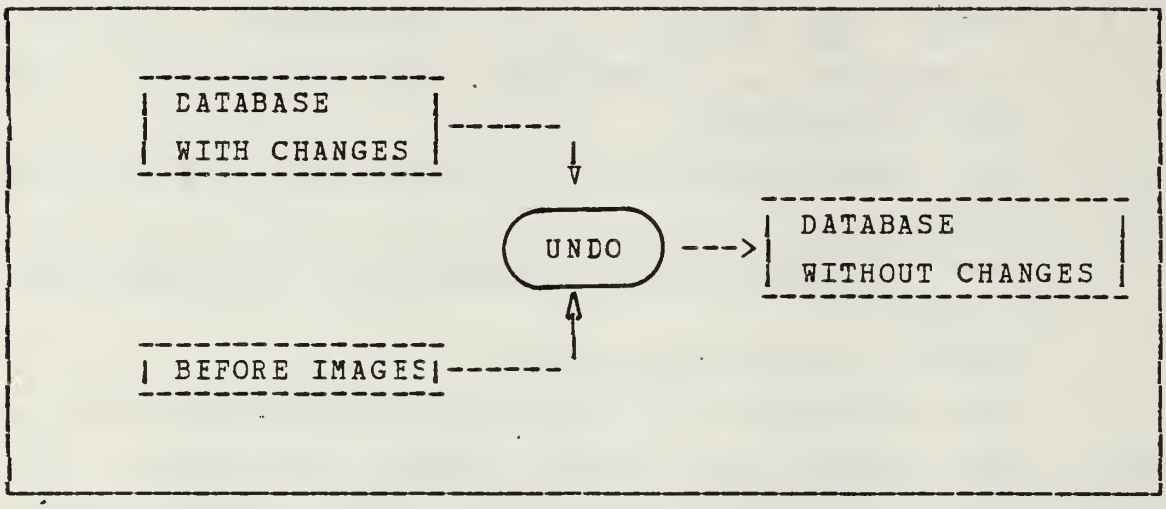


Figure 10.2 UNDO Transaction Procedure.

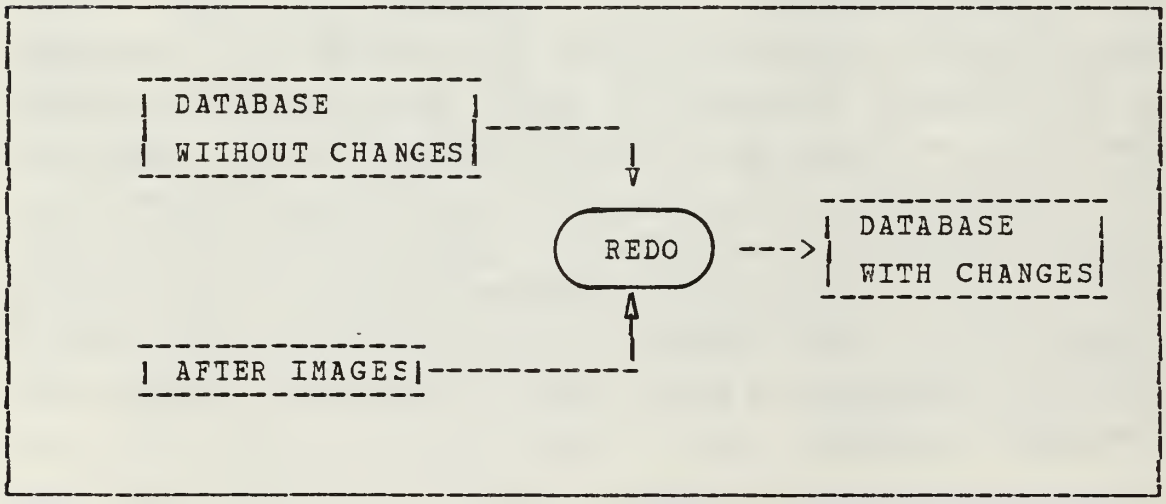


Figure 10.3 REDO Transaction Procedure.

To UNDO a transaction, the log must contain a copy of every database record before it was changed. Such records are called before images. By applying before images to the database an UNDO procedure is performed. To REDO a transaction the log must contain a copy of every database record after it was changed. These records are called after images.

By applying after images to the database a REDO procedure is performed. Possible data-items of a transaction log are shown in Figure 10.4.

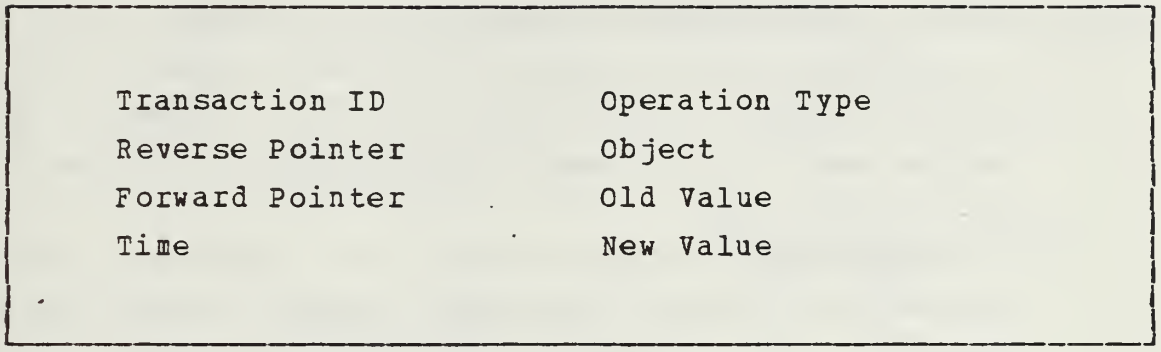


Figure 10.4 Data-items of a Log Record.

For identification purposes each log transaction has a unique ID. All images are linked together with a double-linked list. These forward and backward links can be used by the recovery manager to locate all records for a particular transaction. Other data-items are: the time of the action, the type of the operation (modify,insert, etc.), the object such as record type and identifier, and the old and new values.

5. Write-ahead log

There is an interval between writing a change to the database's stable storage and writing the log record representing that change. These are two distinct operations. This fact introduces two questions: What happens if a failure occurs in the interval between these two operations? What should be done to avoid improper applications?

Suppose that in fact such a failure does occur, so that only one of the writes (the first operation) is

performed and the other is lost. If the performed operation is the database write, there will be changes in the database that are not recorded in the log so the UNDO process is not possible for these changes. It is obvious that, for safety, the log record should always be written first. Therefore we can define the write-ahead log protocol as follows:

1. A transaction is not allowed to write a record to the stable storage of the database until at least the before image of the log record has been written to the physical log.
2. A transaction is not allowed to complete COMMIT processing until both the before images and the after images of all log records for the transaction have been written to the physical log.

If a failure occurs, a change may be recorded in the log and not in the database. In this case, the recovery manager may attempt to undo changes that have not yet occured. This is not a problem, because the recovery manager will only be placing before images in the database. Records will be replaced by copies of themselves. This is a wasteful operation but not harmful.

C. CONCURRENCY CONTROL

Given a correct state of the database as input, a correct transaction will produce a correct state of the database as output. Even if all transactions are individually correct, however, it is possible in a multiuser system for transactions that execute concurrently to interfere with one another in such a way as to produce an overall result that is not correct. As an example of that kind of interference, we will consider the "lost update" problem.

1. Concurrent Update (Lost Update) Problem

The lost update problem can be represented as shown in Figure 10.5.

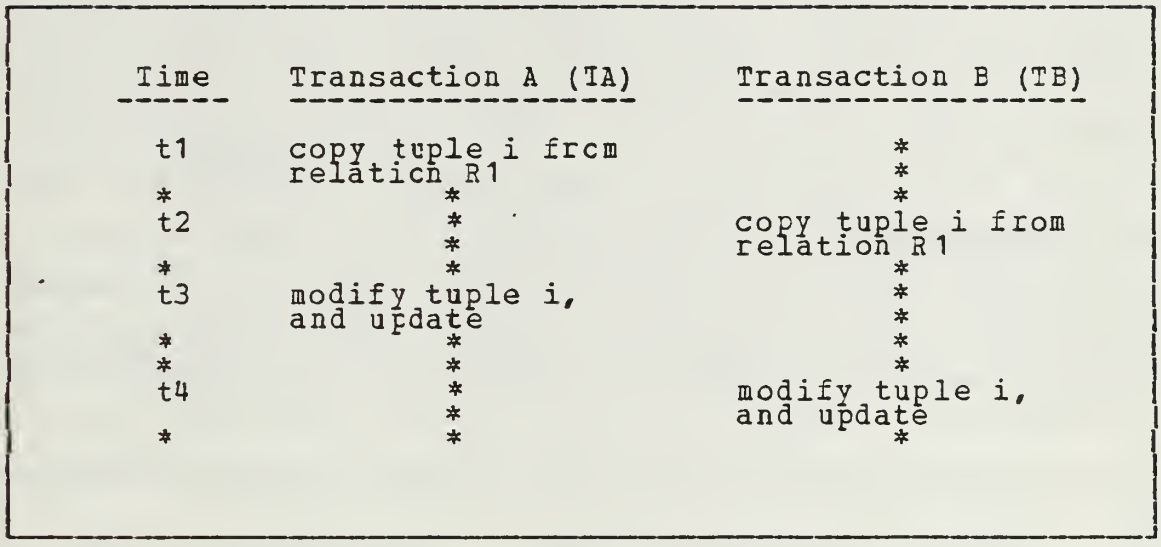


Figure 10.5 Lost Update Problem.

Transaction A is intended to change some field F in tuple i; lets say will double the value of field F. Transaction B is intended also to double the value of that same field. Thus, if the initial value of that field is 2, then running the two transactions one at a time, without concurrency, will produce a final result of 8. However, the particular concurrent execution sequence shown in Figure 10.5 produces a final result of 4. That particular execution sequence is therefore incorrect. In this situation, we can say that TA's update is lost because TB overwrites it.

2. Resource Locking

The most common method of concurrency control is to use locks. One lock is maintained for each user. The term

user refers to the user of DBMS, not necessarily the system user. Thus a user can be either a person using the DBMS query/update facility via a terminal, or an application program that calls upon the DBMS for service. A program obtains all such locks before making any updates. Concurrency control must ensure that at most one program gets the lock for one database part. Hence, if a program wishes to move a locked item to the program working storage, it must wait until the previous program releases the lock. The implementation of this process differs from one system to another. In many implementations, user programs include commands to lock the required records before updating them. This can be represented pictorially as shown in Figure 10.6.

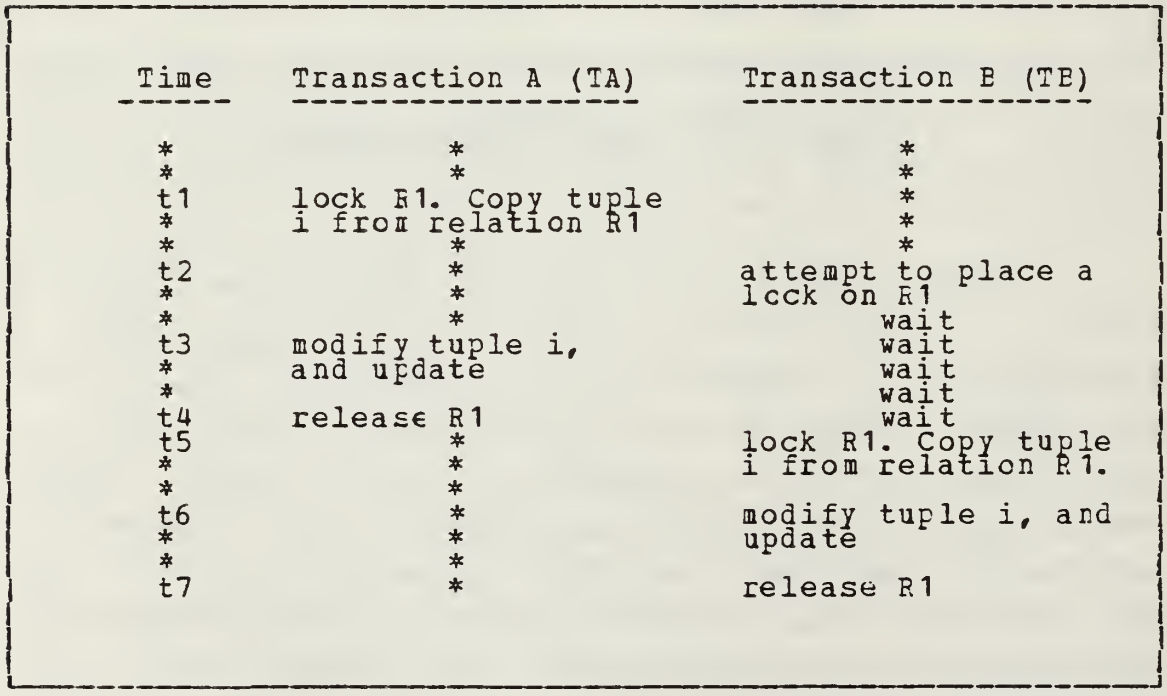


Figure 10.6 Resource Locking.

It is seen that transaction B is now made to wait at time t2, because its request for a lock on R1 at that time conflicts with the lock already held on R1 by transaction A. Transaction B resumes after transaction A releases its lock. This kind of lock mechanism will provide a correct final result for those two transactions. Lost update problems can be solved by the lock mechanism.

3. Deadlock

Locks can introduce the problem commonly known as deadlock. A deadlock occurs when two transactions, say TA and TB, each places locks on relations, say R1 and R2 respectively, and then each transaction attempts to place a lock on the others already locked relation. The order of processing can be as shown in Figure 10.7.

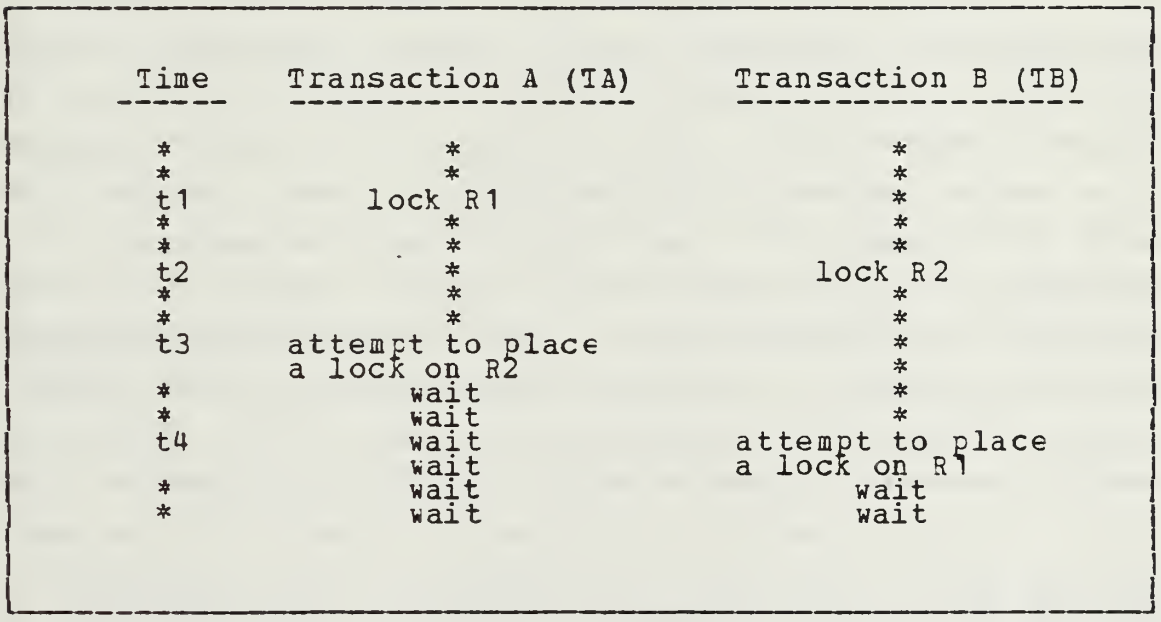


Figure 10.7 Deadlock Problem.

Both transactions are then waiting for each other to release a lock. In the database environment, the usual step to resolve or "to break" the deadlock is to rollback one of the programs. Breaking a deadlock consist of choosing a "victim", one of the deadlocked transactions; and rolling it back. The victim is not necessarily the transaction that actually caused the deadlock; it may be the one holding the fewest locks, or the one that was most recenly started, or the one that has made the fewest updates. The rollback process involves the following jobs:

1. Terminate the transaction, victim, and undo all of its updates.
2. Release all the locks of the transaction; resources are now allocated to other transactions.

4. Lock Granularity

So far, in our examples, we assumed that the unit of locking is the individual record. However, the level of the lock can be different in different applications, or in different DBMSs. Locks, at the highest level, can be applied to an entire database. This strategy is used by DBMS products that invoke the lock for a short time during the processing of a single database request. Locks can also be applied, at the lowest level, to a specific field within an individual record. In between these extremes, locks can be placed on records, on pages or blocks, and on files. As usual, there are tradeoffs among these alternatives. A lock of the entire database is simple for the DBMS to manage. However, throughput may be slow because of less concurrency. On the other hand, locks of small granularity will be complex to manage but throughput will tend to be faster because of more concurrency. The choice among alternatives depends on requirements.

D. DATABASE SECURITY

The security in database environment is protection of the database against unauthorized disclosure, alteration, or destruction. The subject of database security has many different aspects and approaches, such as physical protection, hardware controls, using passwords, or using authorization tables. Here we are concerned primarily with restricting certain users so they are allowed to access and/or modify only a subset of the database.

Good security means that people have access to the data that they need to accomplish their job function, and no more. Job functions vary, and for that reason data access authorizations will vary. A table called authorization rules is used for that purpose, and was developed by Fernandez, Summers, and Wood in [Ref. 22:p. 5].

Authorization rules are compiled and stored in the system dictionary. First, these rules will be entered into the system, then they will be enforced. The authorization rules compiler and the corresponding enforcement mechanism together make up the security subsystem.

In the application environment it is convenient to use a matrix for authorization rules. The matrix is called an authorization matrix in which rows correspond to users and columns correspond to data objects. The entry A[i,j] represents the set of authorization rules that apply to user i with respect to data object j. An example of an authorization matrix is shown in Figure 10.8.

Sophistication of the security subsystem can be measured by the granularity of the objects. For example, some DBMS systems support authorization only at the level of whole relations, others permit authorization at the level of individual fields. In our example authorization is based on the

	DATA OBJECT1 (OFFICER relation)	DATA OBJECT2 (UNIT relation)	DATA OBJECT3 (COURSES relation)
USER1 (Brown)	ALL	ALL	ALL
USER2 (John)	NONE	NONE	NONE
USER3 (Personel Office)	ALL	READ	READ UPDATE
USER4 (Prog-3)	NONE	READ	READ
USER5 (Education Office)	READ UPDATE	READ	ALL

Figure 10.8 An Example for Authorization Matrix.

names of objects and not on their value. This is called value-independent control. In this schema, the system can enforce the controls without having to access the data objects themselves. It is also possible to provide value-dependent control in that we can extend the entries in the matrix to include an optional access predicate. For example, the entry

```
SELECT *
FROM OFFICER
WHERE RANK = 'CAPT'
```

might be used to allow SELECT access to some officers and not others.

Authorization rules can also specify that certain field combinations are prohibited, even though the individual fields within the combination may be accessible. It is also necessary to control access to programs. Moreover, it is important to control access to the authorization matrix itself.

XI. CONCLUSIONS

Information is a basic resource, like people or money, for an enterprise, and it should have a professional management group that is responsible for its effective use throughout the enterprise. For achieving this task, a new staff function called information resource management (IRM) has been proposed. This function, in most cases, should establish policies and procedures to guide users, system developers, and managers so that their decisions will be consistent and compatible and employ the best in currently available technology. In a DBMS, this function is referred to as Database Administration (DBA). [Ref. 15:pp. 168-183]

On the other hand, the personnel administration function for managers of an organization must have complete control over evaluating, assigning, and firing their own employees. In order to perform this task satisfactorily and effectively, the managers have to make their own decisions very accurately. Sometimes, they are forced to make such decisions in a short period of time. Those factors in a powerful personnel management can be provided by having a well-designed personnel database and a suitable DBMS.

James P. Fry and Edgar H. Sibley state in their 1976 paper [Ref. 23] that the objectives of database management are:

- to make an integrated collection of data available to a wide variety of users (data availability),
- to provide for quality and integrity of the data (data quality),
- to insure retention of privacy through security measures within the system (privacy and security),

-to allow centralized control of the database which is necessary for efficient data administration (management control), and

-to provide a high degree of data independence.

Considering those major objectives and some advantages such as simplicity, ease of use, data independence, and theoretical foundation, the relational database model has been found to be convenient in designing such a personnel database system. The other database models are more complex and more difficult to implement.

After the organization's requirements are understood, the process usually begins by choosing the data model that seems most appropriate and then proceeding to a detailed evaluation of only the available DBMS products that support the selected model. This is the problem of choosing a DBMS. Several committees are working on this problem such that all DBMS's provide the same functions and the same interfaces will be standard. In this thesis, the ORACLE DBMS is used to show the implementation stage of the personnel database.

In conclusion, it is useful to emphasize that it is not only important to design an efficient database for an enterprise but also it is required to maintain and develop the database by permanently monitoring its performance to maximize efficiency as a final operational responsibility of the database administration (DBA).

APPENDIX A
SEMANTIC DATABASE DESIGN

The detailed description of the Semantic Database Model (SDM) design for the Personnel Database which is mentioned in Chapter V is shown below.

```
OFFICER
  description:  All officers who are on active-duty.
  member attributes:
    Military_ID
      description: A unique number for each officer
      value class: MID
      mandatory
      not changeable

    Rank
      value class: RANK

    Date_of_promotion
      value class: DATE

    Name
      value class: PERSON_NAMES

    Birth_date
      value class: DATE

    Beginning_date_to_active-duty
      description: Date of first day of being on
                   active-duty.
      value class: DATE

    Native_country
      value class: COUNTRY

    Sex
      value class: SEX

    Marital_status
      value class: MARITAL_STATUS

    Number_of_children
      value class: INTEGERS

    Permanent_address
      value class: ADDRESS

    Current_address
      value class: ADDRESS

    Primary_branch
      description:
      value class: BRANCHES

    Secondary_branch
      description:
      value class: BRANCHES

    Academic_education
      value class: ACADEMIC_MAJOR
      match : Academic_branch/Academic_degree of
              ACADEMIC_MAJOR on AID.
      multivalued
```

Figure A.1 SDM Design for Personnel Database.

```
    Military_education/courses
      value class: MILITARY_EDUCATION/COURSES
      match : Course/Military_school_code of
              MILITARY_EDUCATION/COURSES on MEID.
      multivalued

    Health_condition
      value class: MEDICAL_INFO
      match : General_health_status of
              MEDICAL_INFO on HID.

    Foreign_language_capability
      value class: FOREIGN_LANGUAGE
      match : Foreign_language of
              FOREIGN_LANGUAGE on FID.
      multivalued

    Unit_assigned
      description: Units which the officer has been
                   assigned until current date.
      value class: UNIT
      inverse    : Officer_assigned
      multivalued

  identifiers:

    Military_ID


ACADEMIC_MAJOR

  description: Type of academic branch,the degree
               earned for that branch, location
               and name of the universty which
               the officer attended.

  member attributes:

    Academic_branch
      description: Branch such as Computer Science,
                   Electrical Engineering.
      value class: ACADEMIC_BRANCHES

    Academic_degree
      description: Degree such as Bachelor of
                   Science, Master of Science,
                   Engineering, Doctorate.
      value class: ACADEMIC_DEGREES

    AID
      description: Military_ID of the officer who
                   earned that degree.
      value class: MID
      mandatory

    Date
      description: Date at which the degree earned
      value class: DATE
```

Figure A.2 SDM Design for Personnel Database (cont'd.).

```
    Name_of_university
      value class: UNIVERSITY_NAMES

    Location_of_university:
      value class: COUNTRY

  identifiers:

    Academic_branch + Academic_degree + AID


MILITARY_EDUCATION/COURSES

  description: Information about the military
               school graduated or military course
               attended, location of school, grade
               and date of graduation.

  member attributes:

    Course/Military_school_code
      value class: COURSE/SCHOOL_CODE

    Location
      value class: COUNTRY

    MEID
      description: Military_ID of the officer who
                   attended the school or course.
      value class: MID
      mandatory

    Course/School_title
      value class: COURSE/SCHOOL_TITLES

    Description
      description: Textual explanation of the course
      value class: COURSE/SCHOOL_DESCRIPTION

    Duration
      description: Duration of the military
                   education in weeks.
      value class: INTEGERS

    Date
      description: Graduation date of an officer
                   from the course or school.
      value class: DATE


    Grade
      description: Grade earned for that course
                   or military education.
      value class: COURSE_GRADES

  identifiers

    Course/Military school_code + Location + MEID
```

Figure A.3 SDM Design for Personnel Database (cont'd.).

```
MEDICAL_INFO
  description: Medical information and overall
               medical status for an officer.
  member attributes:
    Medical_report_number
      description: Medical report number of last
                   checking of the officer.
      value class: REPORT_NUMBER
    HID
      description: Military_ID of the officer to
                   whom the information belongs to.
      value class: MID
      mandatory
    Date
      description: Date of the report.
      value class: DATE
    Height
      description: Height of the officer.
      value class: HEIGHT
    Weight
      description: Weight of the officer.
      value class: WEIGHT
    Blood_pressure
      value class: BLOOD_PRESSURE
    Eye_condition
      description: Describes the condition of both
                   eyes of the officer.
      value class: EYE_CONDITION
    Ear_condition
      description: Describes the condition of both
                   ears of the officer.
      value class: EAR_CONDITION
    Internal
      description: Describes the condition of
                   internal organs of the officer.
      value class: INTERNAL_CONDITION
    General_health_status
      description: An overall evaluation of
                   conditions of all body parts.
                   This status is described by some
                   member attribute values of this
                   entity class.
      value class: HEALTH_STATUS
```

Figure A.4 SDM Design for Personnel Database (cont'd.).

```
    identifiers:

      Medical_report_number + HID

  FOREIGN_LANGUAGE

    description: It is used to define the officers
                 fcreign language capability.

    member attributes:

      Foreign_language
        value class: LANGUAGES

      FID
        description: Military ID of the officer who
                     has the language capability.
        value class: MID
        mandatory

      Degree_of_capability
        value class: LANGUAGE_CAPABILITY

    identifiers:

      Foreign_language + FID


  UNIT
    description: Description of a unit. Unit code,
                 unit name,unit categori,location,
                 superior unit,unit status and
                 officers assigned to the unit.

    member attributes:

      Unit_code
        value class: UNIT_CODE
        mandatory
        not changeable

      Name
        value class: UNIT_NAMES

      Unit_category
        description: Organizational level of unit
                     such as corps, brigade,
                     division.

        value class: UNIT_CAT

      Location
        description: Location of unit.

        value class: UNIT_LOCATION
```

Figure A.5 SDM Design for Personnel Database (cont'd.).

```
    Superior_unit
      description: The unit which has command and
                   control of this unit.
      value class: UNIT

    Unit_function
      description: Type of function or service which
                   the unit performs.
      value class: UNIT_FUNC

    Officer_assigned
      description: Officers who are assigned to this
                   unit.

      value class: OFFICER
      inverse    : Unit_assigned
      multivalued

  identifiers:

    Unit_code


ASSIGNMENT_REQUEST

  description: The request which is made by any unit,
               about officers who have certain
               specifications fit for a specific
               position to be assigned.

  member attributes:

    Unit_code
      description: The unit who is issued the
                   request for assignment.
      value class: UNIT_CODE
      mandatory
      not changeable

    Request_number
      description: A number which is given by the
                   unit who is issued the request.
      value class: REQUEST_NO
      mandatory
      not changeable

    Date
      value class: DATE

    Rank
      description: Rank of the officer who is
                   requested for assignment.
      value class: RANK

    Primary_branch_requested
      value class: BRANCHES
```

Figure A.6 SDM Design for Personnel Database (cont'd.).

```
    Secondary_branch_requested
      value class: BRANCHES

    Academic_major_requested
      description: Academic major and degree for
                   this assignment
      value class: ACADEMIC_MAJOR
      multivalued

    Military_course/education_requested
      description: Military education and/or course
                   which is needed for this
                   assignment.
      value class: MILITARY_EDUCATION/COURSES
      multivalued

    Medical_status
      description: Lowest value for medical status
                   which is needed for this
                   assignment.
      value class: HEALTH_STATUS

    Number_of_person
      description: Number of officer requested with
                   this assignment request.
      value class: INTEGERS

  class attributes:

    Number_of_requests
      description: The number of requests that
                   issued in the current year.
      derivation: Number of members in this class
                  which Date= current year.

  identifiers:

    Unit_code + Request_number
```

Figure A.7 SDM Design for Personnel Database (cont'd.).

```
MID
  interclass connection: subclass of STRINGS where
     format is 5 digit numbers

RANK
  interclass connection: subclass of STRINGS where
     specified

DATE
  interclass connection: subclass of STRINGS where
     format is:
     month: number where ≥1 and ≤12
     "-"
     day: number where integer and ≥1 and ≤31
     "-"
     year: number where integer and ≥1900and≤2000
     where (if(month =4 or =5 or =9 or =11) then
           day ≤ 30) and (if month =2 then day ≤29)
     ordering by year, month, day

PERSON_NAMES
  interclass connection: subclass of STRINGS

COUNTRY
  interclass connection: subclass of STRINGS where
     specified

SEX
  interclass connection: subclass of STRINGS where
     format is 1 character: m, f

MARITAL_STATUS
  interclass connection: subclass of STRINGS where
     format is 1 character: S, M, D,

ADDRESS
  interclass connection: subclass of STRINGS

BRANCHES
  interclass connection: subclass of STRINGS where
     specified

ACADEMIC_BRANCHES
  interclass connection: subclass of STRINGS where
     specified

ACADEMIC_DEGREES
  interclass connection: subclass of STRINGS where
     values are: BA, BS, MA, MS, ENG, PhD

UNIVERSITY_NAMES
  interclass connection: subclass of STRINGS

COURSE/SCHOOL_CODE
  interclass connection: subclass of STRINGS where
     format is 5 characters

COURSE/SCHOOL_TITLES
  interclass connection: subclass of STRINGS
```

Figure A.8 Domains of Attributes.

```
COURSE/SCHOOL_DESCRIPTION
  interclass connection: subclass of STRINGS

COURSE_GRADES
  interclass connection: subclass of STRINGS where
     format is 2 characters

REPORT_NUMBER
  interclass connection: subclass of STRINGS where
     format is 6 digit number

HEIGHT
  interclass connection: subclass of STRINGS where
     format is positive integer

WEIGHT
  interclass connection: subclass of STRINGS where
     format is  positive integer

BLOOD_PRESSURE
  interclass connection: subclass of STRINGS where
     specified

EYE_CONDITION
  interclass connection: subclass of STRINGS where
     specified

EAR_CONDITION
  interclass connection: subclass of STRINGS where
     specified

INTERNAL_CONDITION
  interclass connection: subclass of STRINGS where
     specified

HEALTH_STATUS
  interclass connection: subclass of STRINGS where
     format is 2 digit number

LANGUAGES
  interclass connection: subclass of STRINGS where
     specified

LANGUAGE_CAPABILITY
  interclass connection: subclass of STRINGS where
     format is 1 digit number

UNIT_CODE
  interclass connection: subclass of STRINGS where
     specified

UNIT_NAMES
  interclass connection: subclass of STRINGS

UNIT_CAT
  interclass connection: subclass of STRINGS where
     format is 3 characters: COR, DIV, BRI, REG,
     BTE
```

Figure A.9 Domains of Attributes (cont'd).

```
UNIT_LOCATION
  interclass connection: subclass of STRINGS where
     specified

UNIT_FUNC
  interclass connection: subclass of STRINGS where
     format is 6 characters

REQUEST_NO
  interclass connection: subclass of STRINGS where
     format is 6 digit number
```

Figure A.10 Domains of Attributes (cont'd).

APPENDIX B

SAMPLE RELATIONS FOR PERSONNEL DATABASE

This Appendix shows sample relations of the Personnel Database, which are used to implement. The implementation of this database is based on these sample relations.

1. Relation OFFICER:

```
UFI> SELECT *
  2  FROM OFFICER;

    MID RANK ONAME      SEX PRI-BRAN SEC-BRAN OTHERS
------- ---- --------- --- -------- -------- --------
  27363 capt Johnson   m   artill   pilot
  12239 maj  Hernandez m   inftry   spefc
  32458 1lt  Robbins   f   airdef   ado
  43596 2lt  Smith     m   medic    pilot
  10999 lcol Brown     m   inftry   ado
  35768 1lt  Greenberg m   signor   pilot
  29364 capt James     m   milenq   spefc
  16745 maj  Leighton  m   financ   ado
  10792 col  Stone     m   ordnan   artill

9 records selected.
```

2. Relation UNIT:

```
UFI> SELECT *
  2  FROM UNIT;

UCODE UNAME           UCAT ULOC           SUP-UNIT UFUNC
----- --------------- ---- -------------- -------- -----
01DIV 1st Inf Div     div  Ft.Riley,KS    09ARM    comba
09BRG 9th Art Brg     brg  Ft.Riley,KS    01DIV    comsu
64DEP 64th Ord Depot  dep  Ft.Riley,KS    01DIV    comse
12HOS 12th Fd Hosp    hosp Ft.Riley,KS    01DIV    comse
02AVI 2nd Avia Unit   reg  Ft.Gordon,GA   08DIV    comba
07SGP 7th Spe Fcs Gp  bt   Ft.Bragg,NC    82DIV    comba
20ABT 20th Airdef Bt  bt   Ft.Hood,TX     02DIV    comsu
03ENB 3rd Eng Bn      bn   Ft.Hood,TX     02DIV    comba
02DIV 2nd Inf Div     div  Ft.Hood,TX     07ARM    comba

9 records selected.
```

3. Relation A_EDUCATION:

```
UFI> SELECT *
  2  FROM A*EDUCATION;

ABRAN ADEG      AID UNIV         GDATE
----- ---- ------- ----------- ---------
math  BS     27363 UCLA,CA     06-AUG-75
law   BA     12239 Purdue,IN   30-JUL-70
ee    BS     32458 Seattle,WA  21-SEP-80
dent  BS     43596 Berkeley,CA 25-MAY-80
mngt  BA     10999 Loyola,IL   01-JUL-68
cs    MS     10999 NPGS,CA     07-DEC-73
ee    BS     35768 Rice,TX     06-MAY-80
cons  BS     29364 MIT,MA      09-DEC-78
ee    MS     29364 Ohio,OH     10-JUL-85
mngt  BA     16745 Cornell,NY  30-SEP-75
me    BS     10792 Rutgers,NJ  31-AUG-65
ee    MS     10792 NPGS,CA     01-MAR-70

12 records selected.
```

4. Relation M_EDUCATION:

```
UFI> SELECT *
  2  FROM M*EDUCATION
  3  ORDER BY CCODE&,CGRADE;

CCODE& MEID CGRADE CDATE
------ ----- ------ ---------
AA102  43596 A-     12-JUL-84
AD002  32458 A      23-NOV-82
AS003  27363 B+     13-APR-77
CS302  32458 B+     26-OCT-84
CS509  10999 A-     31-JAN-77
HS706  43596 A      22-JUN-82
IA076  16745 A-     22-NOV-76
IS005  10999 A      11-OCT-70
IS005  12239 A-     30-SEP-72
OC092  10792 A-     01-DEC-69
SS002  35768 B+     26-FEB-82

11 records selected.
```

5. Relation M_COURSES:

```
UFI> SELECT *
  2  FROM M_COURSES;

CCODEB CLOCB           CTITLE     CDESC                          CDUR
------ --------------- ---------- ------------------------------ -------
AS003  Ft.Sill,OK      artillery  Army Field Artillery School       48
AA102  Ft.Rucker,AL    aviation   Army Aviation School             52
AD002  Ft.Bliss,TX     airdef     Army Air Defense School          36
CS302  Monterey,CA     adp        Adp Officer Course               13
CS509  Ft.Harrison,IN  adp        Adp Officer Course               13
IA076  Ft.Harrison,IN  admin      Ins.for Administration           30
IS005  Ft.Benning,GA   infantry  Army Infantry School             48
HS706  Ft.Houston,TX   health     Aca. of Health Sci.              96
OC092  Aberdeen,MD     ordchem    Army Ord.and Chem.School         24
SS002  Ft.Gordon,GA    signal     Army Signal School               50

10 records selected.
```

6. Relation LANGUAGE:

```
UFI> SELECT *
  2  FROM LANGUAGE;

NLANGUAGE      FID LDEGREE
--------- ------- -------
german       27363       6
french       12239       4
russian      12239       7
korean       15745       2
german       35768       5
turkish      32458       4

6 records selected.
```

7. Relation MEDICAL:

```
UFI> SELECT *
  2  FROM MEDICAL;
```

REPNO	HID	RDATE	EYECOND	EARCOND	HSTAT	OTHERS
98381-7	12239	30-NOV-81	0	0	0	
13282-5	32458	01-MAR-82	12	11	2	
24582-6	43596	30-AUG-82	0	11	1	
37584-1	27363	07-JUN-84	11	0	1	
48384-3	35768	31-AUG-84	10	0	1	
12885-4	29364	09-MAR-85	0	0	0	
20985-6	16745	14-APR-85	11	11	2	
24580-0	10999	17-NOV-80	22	0	2	
37580-8	10792	06-DEC-80	45	22	6	
25681-2	27363	04-OCT-81	0	0	0	

```
10 records selected.
```

8. Relation ASSIGNMENT:

```
UFI> SELECT *
  2  FROM ASSIGNMENT;
```

AMID	AUCODE	ORDERNO	ASGDATE
10792	098RG	038165-1	01-SEP-66
10999	01DIV	327467-8	24-SEP-69
16745	01DIV	456277-3	29-DEC-77
27363	098RG	321578-7	12-SEP-78
29364	64DEP	593879-5	08-MAR-79
12239	02DIV	491373-6	30-OCT-73
32458	20ABT	482683-2	10-JAN-83
43596	12HOS	321782-4	11-AUG-82
35768	01DIV	152282-9	01-APR-82
10792	02DIV	324871-5	11-JAN-71
10999	02DIV	118273-3	30-AUG-73
10792	64DEP	324678-1	15-APR-78

```
12 records selected.
```

9. Relation ASGREQ:

```
UFI> SELECT *
  2  FROM ASGREQ;

R-UCODE REQNUM    REQDATE   R-RANK R-PRIBR R-SECBR R-ACABR R-MILED R-MSTAT
------- --------- --------- ------ ------- ------- ------- ------- -------
NUMOFPERS
---------

12MOS   031084-3  30-DEC-84 maj    financ  ado     mngt    IA076         4
        2

20ABT   922185-7  06-MAR-85 capt   artill  pilot   ee      AS003         2
        1

20SGP   327685-4  27-APR-85 maj    inftry  spefc   law     IS005         0
        1
```

LIST OF REFERENCES

1. Hawryszkiewycz, I.T., Database Analysis and Design, SRA, 1984

2. McGee, W.C., "Database Technology", IBM J.Res.Develop., Vol.25,No.5, September 1981, pp. 505-518

3. Date, C.J., An Introduction to Database Systems, Volume I, Addiscn Wesley, February 1982

4. Engles, R.W., A Tutorial on Data Base Organization, Annual Review in Automatic Programming, Volume 7, The Pergamon Press, 1974

5. Teorey, Toby J.and Fry, James P., Design of Database Structures, Prentice-Hall, 1982

6. Krcenke, D., Database Processing: Fundamentals, Design, Implementation, SRA, 1983

7. Atre, S., Database: Structured Techniques for Design, Performance, and Management with Case Studies, A Wiley Series, 1980

8. Tsichritzis, D.C. and Klug, A., "The ANSI/X3/SPARC DBMS Framework: Report of The Study Group on Data Base Management Systems.", Information Systems 3, 1978

9. Ullman, Jeffrey D., Principles of Database Systems, Ccmputer Science Press, 1982

10. Hammer, M. and McLeod, D., "Database Description with SDM: A Semantic Database Model", ACM Transactions on Database Systems, Vol. 6, No. 3, September 1981

11. Fagin, R., "A Normal Form for Relational Databases That is Based cn Domains and Keys." In Transactions on Database Systems, Vol.6,No.3, September 1981, pp. 387-415

12. Codd,E.F., A Data Base Sublanguage Founded on the Relational Calculus., Proceedings of the 1971 ACM SIGFIDET Conference on Data Description, Access and Control, San Diego, pp. 35-68

13. Held, G.D., Stonebraker, M.R., and Wong, E., INGRES-A Relational Data Base System, Proc. NCC44, May 1975, pp. 409-416

14. Astrahan, M.M., et.al., "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol.1, No.2, June 1976, pp. 97-137

15. Goldstein, R.C., Database: Technology and Management, John Wiley & Sons, 1985

16. Beeri, C., Bernstein, P.A., Goodman, N., A Sophisticate's Introduction to Database Normalization Theory, Proceedings of the Fourth International Conference on Very Large Data Bases, West Berlin, 1978, pp. 113-124

17. Kent, W., "A Simple Guide to Five Normal Forms in Relational Database Theory", Communications of the ACM, Vol.26, No.2, February 1983, pp. 120-125

18. Fagin, R., "Multivalued Dependencies and a New Normal Form for Relational Databases", ACM Transactions on Database Systems, Vol. 2, No. 3, September 1977, pp. 262-278

19. Aho, A.V., et. al., "The Theory of Joins in Relational Databases", ACM Transactions on Database Systems, Vol. 4, No. 3, 1979, pp. 297-314

20. Codd, E.F., "Relational Database: A Practical Foundation for Productivity", Communications of the ACM, Vol.25, No.2, February 1982

21. Gray, J., et. al., "The Recovery Manager of the System R Database Manager", Computing Surveys, Vol. 13, No. 2, June 1981

22. Fernandez, E.B., Summers, R.C.,and Wood, C., Database Security and Integrity, Addison-Wesley, 1981

23. Fry, J.P., Sibley, E.H., "Evolution of Data-Base Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976, pp. 7-42

BIBLIOGRAPHY

Banerjee, J., Hsiao, D.K., DBC Software Requirements for Supporting Relational Databases Naval Postgraduate School, July 1983

Date, C.J., An Introduction to Database Systems, Volume II, Addison Wesley, 1983

Honkanen, P.A., Concurrency Control and Integrity Preservation for the 'RIM' DBMS Database, Spring 1984

Hsiao, D.K., et. al., The Implementation of a Multi-Backend Database System (MDBS): Part III - The Message-Oriented Version with Concurrency Control and Secondary-Memory-Based Directory Management, Naval Postgraduate School, March 1983

Mohan, C., An Overview of Recent Data Base Research, Database, Fall 1978

ORACLE Corporation, ORACLE SQL/UFI Reference Manual - Version 4.0, ORACLE Corporation, June 1984

ORACLE Corporation, ORACLE Terminal User Guide - Version 3.1, Relational Software, Inc., March 1983

Stocker, P.M., Gray, P.M.D., and Atkinson, M.P., Databases-Role and Structure, Cambrigde University Press, 1984

Wiederhold, G., Database Design, McGraw-Hill, Inc., 1977

INITIAL DISTRIBUTION LIST

No.		Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
4.	Professor S.H. Parry, Code 55Py Department of Operations Research Naval Postgraduate School Monterey, California 93943-5100	2
5.	Dr. D.K. Hsiao, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5100	1
6.	Curricula Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5100	1
7.	Gnkur. Mu. Elk. Baskanligi Mu. Bilgi Sistemler Dairesi Gnkur./Ankara, TURKEY	4
8.	K.K.K. Mu. Elk. Baskanligi KOKCBI Subesi K.K.K./Ankara, TURKEY	4
9.	Kara Harp Akademisi Komutanligi Kutuphane Ayazaga/Istanbul, TURKEY	1
10.	Kara Harp Okulu Komutanligi Kutuphane K.H.O./Ankara, TURKEY	1
11.	Department Chairman Department of Computer Science Middle East Technical University Ankara, TURKEY	1
12.	Department Chairman Department of Computer Science Hacettepe University Beytepe/Ankara, TURKEY	1
13.	Department Chairman Department of Computer Science Bogazici University Yenikoy/Istanbul, TURKEY	1

14. Enb. Bora Buyukoner 4
Gnkur. Mu. Elk. Baskanligi
Mu. Bilgi Sistemler Dairesi
Gnkur./Ankara, TURKEY

15. Yzb. Yucel Ozin 4
K.K.K. Mu. Elk. Baskanligi
KOKOBI Subesi
K.K.K./Ankara, TURKEY

214095

Thesis
B95192 Buyukoner
c.1 Design and implementation of a personnel database.